ネットワーク接続された組込みシステムの拡張性に関する研究

Studies on Extensibility of Networked Embedded Systems

寺 岡 秀 敏

2019 年 2 月

内容梗概

近年，あらゆるモノがインターネットに接続される Internet of Things（IoT）システムが急速に拡大しつつある．IoT システムは，組込みシステムをインターネット経由でクラウドに接続しシステムとして構成する，ネットワーク接続された組込みシステムの形態の一つである．

ネットワーク接続された組込みシステムとしては，携帯電話などの通信機器の他にも，ホームネットワークシステムや自動車システムなどがあり，これらがインターネットに接続して IoT 化し，スマートハウスやコネクテッドカーを構成する．IoT システムの構成としては，携帯電話やホームルータを介して外部に接続するゲーム機のように，その構成要素である組込み機器が論理的に直接インターネットに接続する形態と，スマートハウスやコネクテッドカーのように，ローカルネットワークに接続した複数の組込みシステム（ノードと称す）からなるシステムがゲートウェイを介して接続する形態がある．

IoT 化したシステムの特長の一つは，組込みシステムから収集した多量のデータに基づく改善や最適化を，実システムにフィードバックする枠組みである．

このような枠組みによるシステム拡張の手段として，携帯電話やゲーム機など前者のシステムでは，インターネット経由でファームウェアやアプリケーションを配布・インストールすることでソフトウェアを再構成（更新）し，セキュリティの対策やユーザへの新たな価値の提供が行われている．

一方で，今後は，ノードのインターネット接続に制限がある後者のシステムや，インターネットに接続しないシステムの拡張も重要となる．そこで，本研究では，このようなノードからのインターネット接続に制限があるネットワーク化された組込みシステムに関し，以下の２つの観点におけるシステム拡張に関する課題を明らかにして，組込みシステムの共通課題を考慮しつつ，その解決を示す．

一つ目の観点として，システム拡張のための構成方法における課題の解決について論じる．ソフトウェアによるシステム拡張のためには拡張を考慮したソフトウェア構造が必要である．特に，コントローラがネットワーク上のノードを制御して機能を実現する集中型システムでは，コントローラとなるノードの拡張性を考慮したソフトウェアの構成方法が課題となる．本論文では，集中型システムの例として，ゲートウェイがコントローラの役割を担うホームネットワークシステムについて論じる．具体的には，標準プロトコルである ECHONET Lite ミドルウェアを，アプリケーションの拡張を考慮した OSGi バンドルとして構成することにより，ノードの追加などによるシステム拡張を容易化する方法を示す．

二つ目の観点として，システム拡張を実行する過程における課題の解決について論じる．

ネットワーク化された組込みシステムでは，システムを構成するノードは多様であり，振る舞いやメモリなどのリソースもさまざまである．ソフトウェアの更新によるシステム拡張を実行するに当たっては，更新データの配信や適用といった過程においても，ノードの多様性の考慮が必要となる．本論文では，多様性を持つシステムの例として，複数の Electronic Control Unit（ECU）が連携して機能を実現する自動車システムについて論じる．具体的には，車載ゲートウェイ上で軽量スクリプト言語を動作させることで ECU ごとの更新方法の差異を吸収し，ECU のソフトウェア更新を容易にする方法を示す．また，ネットワーク帯域が限られた自動車システムにおいて，ソフトウェア更新時間を短縮するために，メモリリソースの少ない ECU にも適用可能な差分更新方法を示す．

これにより，今後益々の普及が予想されている IoT システムを含むネットワーク化された組込みシステムにおいて，コストを抑えつつシステム拡張が可能になり，システムの価値向上やセキュリティ等のリスク低減，それによる社会課題の解決に貢献できる．

目次

図目次

表目次

第1章 緒論

1.1. 本研究の背景と目的

近年，あらゆるモノがインターネットに接続される Internet of Things（IoT）が急速に拡大している．パソコンや携帯電話などのインターネット接続端末に加えて，家電や自動車，ビルや工場など様々なものがインターネットへつながることで，IoT デバイスは 2020 年には 300 億に達すると予測されている（図 1-1）[1].

IoT システムは Cyber Physical System（CPS）としての側面も有し，その付加価値をソフトウェアとして具現する物理システムである[2]．このような IoT システム/CPS は，社会課題の解決や新たな価値の創造への貢献が期待されている．

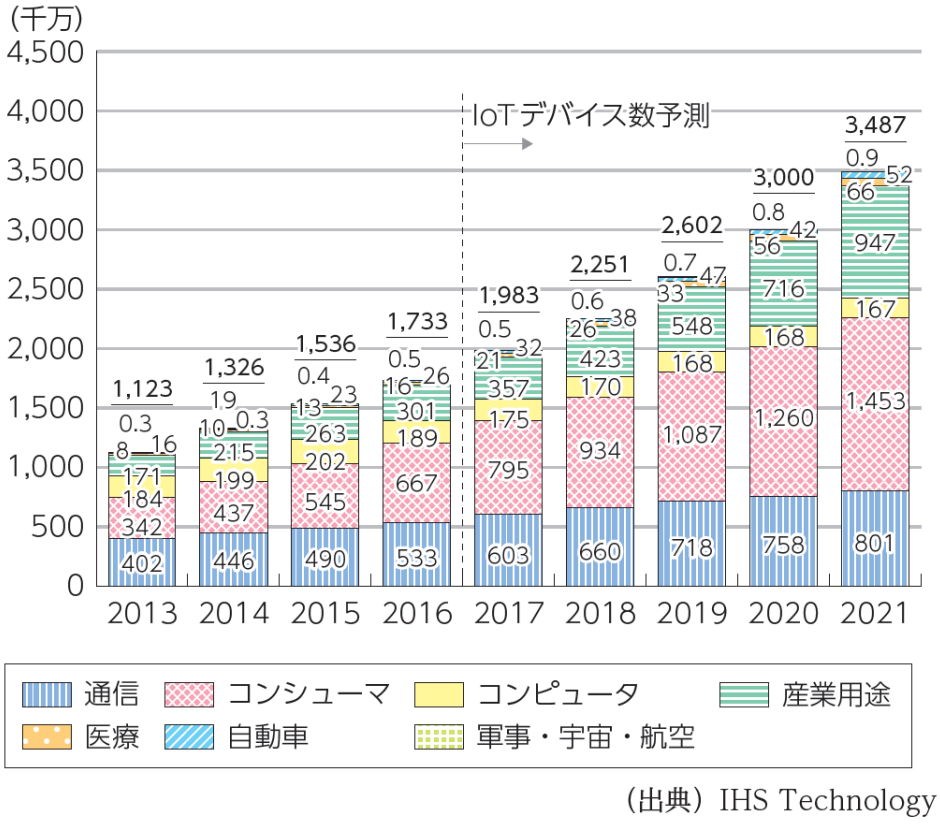


図 1-1　世界の IoT デバイス数の推移及び予測[1]

図 1-1 に示した IoT デバイスの定義は，固有の IP アドレスを持ちインターネットに接続が可能な機器及びセンサーネットワークの末端等として使われる端末等である．このような機器のうち，パソコン等のコンピュータ以外は組込みシステムとして構成される．組込みシステムは，特定用途の機能を実現するマイクロプロセッサーベースのシステム[3]である．従来の家電機器など単独で機能を実現してきた組込みシステムは，ネットワークに接続さ

れて他の組込みシステムと連携し，ホームネットワークシステムや自動車，ビルシステム，工場システムを構成するようになった．さらに，通信技術の進展に伴い，これらがインターネットに接続することで，さらなる機能の向上が進展している．例えば，携帯電話やスマートフォンなどの通信機器，スマートスピーカなどはインターネット越しにサーバと連携して一部の機能を実現している．また，ホームネットワークシステムや自動車，ビルシステム，工場システムはシステムごとインターネットに接続されることで，スマートハウスやホーム／ビルエネルギーマネジメントシステム，コネクテッドカー，スマートファクトリーなどの IoT システムを構成する．

図 1-2 に組込みシステムのネットワーク化の概要を示す．すなわち，単独で動作していた組込みシステム（Embedded Systems）がネットワークに接続され，ネットワーク接続された組込みシステム（Networked Embedded Systems）として機器同士が連携して必要な機能を実現する．さらに，これらのネットワーク接続された組込みシステムが，インターネットに接続し，汎用コンピュータや携帯電話等の単独の組込みシステムとして構成される機器とともに，IoT システムを構成する． すなわち，IoT システムは，ネットワーク接続された組込みシステムの一形態である．

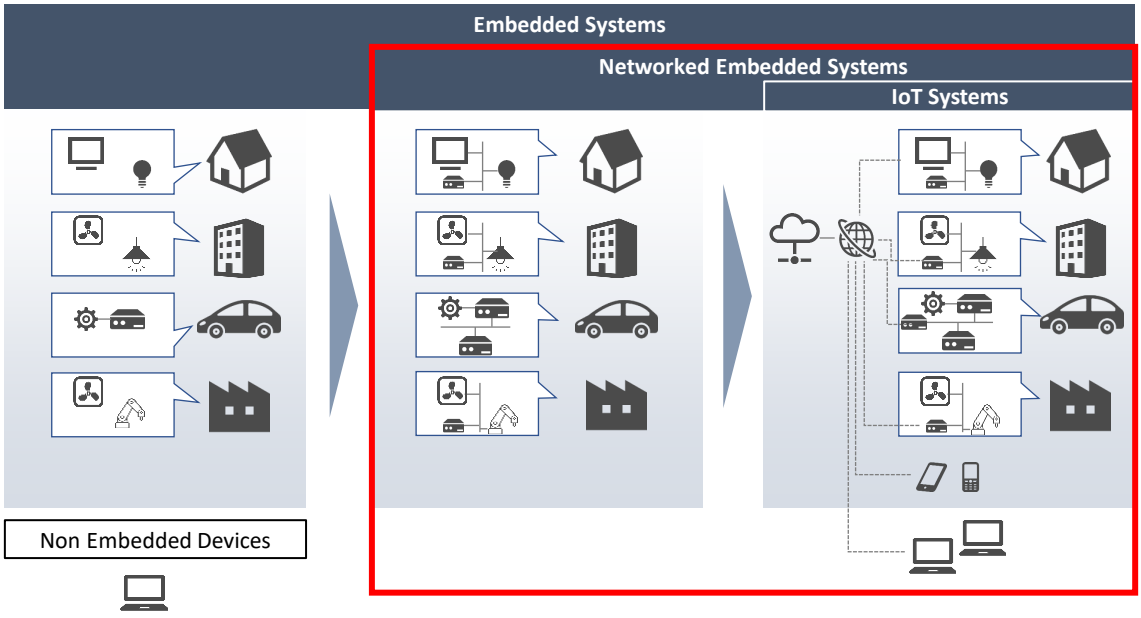
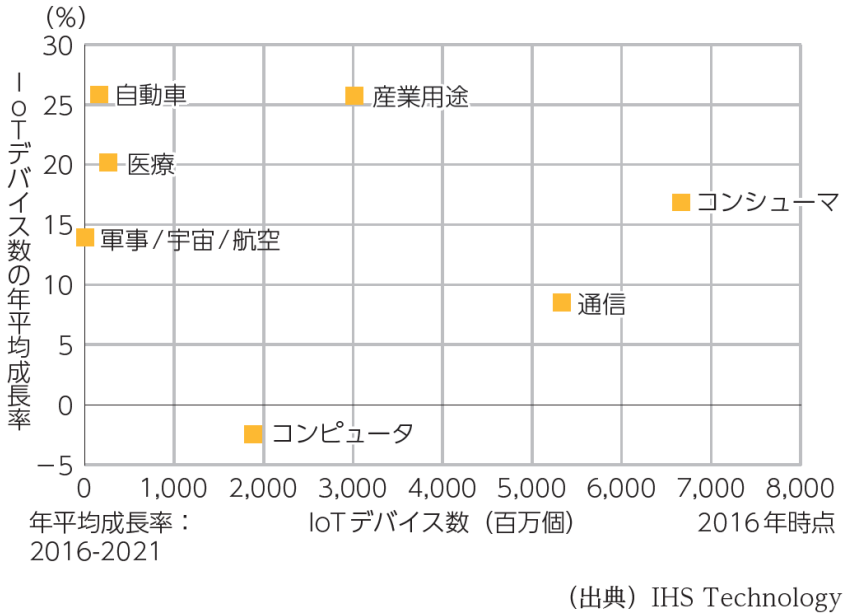


図 1-2 組込みシステムのネットワーク化

図 1-3 に示すように，今後は自動車や産業用途など，複数の組込みシステムがネットワーク接続されて構成されるシステムから成る分野の IoT デバイスの増大が見込まれている[1].



（出典）IHS Technology

図 1-3　分野別・産業別の IoT デバイス数および成長率[1]

1.1.1.　対象システムの特性

(1)　組込みシステム

組込みシステムには一般的に，「専用化されたシステム」，「厳しいコスト要求」とそれに伴う「厳しいリソース制約」と「効率的な開発」，「高い信頼性要求」，「リアルタイム性への要求」といった特性がある[4][5].

家庭で利用される家電や AV 機器から，航空機やプラントに組み込まれる制御機器まで，組込みシステムはそれぞれの用途ごとにハードウェア・ソフトウェアが開発される専用化されたシステムとして開発される．このように特定用途向けに専用に開発が行われる一方，大量に生産される場合が多いため，コストパフォーマンスの高さが非常に重要となる．そのため，プロセッサの処理能力やメモリ量を可能な限り少なくしてデバイスコストを低減することが一般的であるが，これにより組込みシステムで利用可能なリソースは制限される．また，組込みシステムが動作する環境もさまざまであり，低温から高温まで広い温度範囲での動作が要求される．他にも，バッテリで駆動する機器などでは低消費電力も強く要求される．図 1-4 に示すように，ネットワーク接続される機器の待機電力の消費量は機器の増加とともに増大の一途をたどり，2025 年には 46TWh に達すると見込まれている[6].

組込みシステムは，自動車や電車，航空機などの輸送機器をはじめとして，工場やビルなどシステムの制御に利用され，要求された時間内に必要な処理を完了するリアルタイム性が求められる場合が多い．また，このような安全性にかかわる適用先で利用される場合，高い信頼性が求められる．

一方で，近年はデジタル化の進展や製品ラインナップの多様化に伴い，製品の市場投入サイクルの期間が短縮されており，低コストで高信頼な専用機器を短期間で開発するための開発手法，プロセスなどの開発技術も重要な課題となっている[7][8][9].

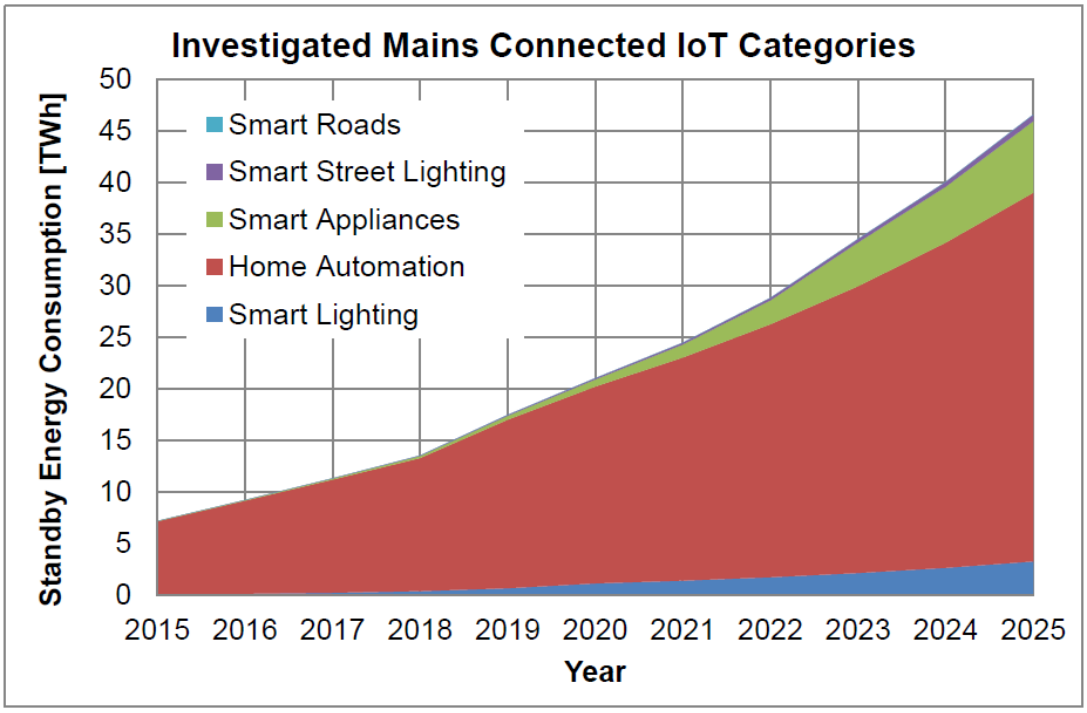


図 1-4 世界の IoT システムのネットワーク接続待機電力量[6]

(2) ネットワーク接続された組込みシステム

ネットワーク接続された組込みシステムでは，ネットワーク上にコントローラ，センサ，アクチュエータが配置され，センサからの入力データに基づき，コントローラがアクチュエータを制御する構成がとられる場合が多い（図 1-5）.

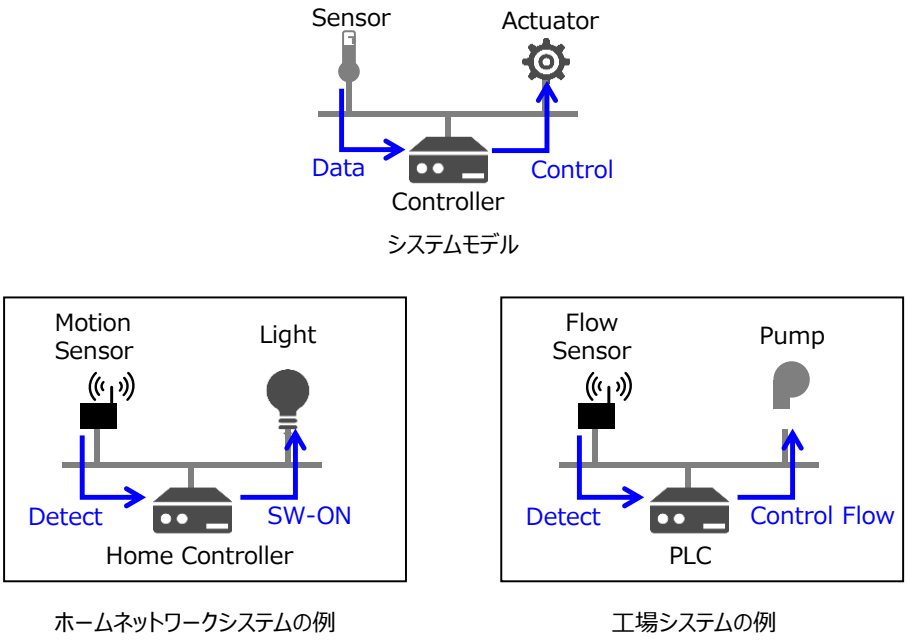


図 1-5 ネットワーク接続された組込みシステム

例えば，ホームネットワークシステムの場合は，人感センサーの検知結果により，照明を点灯するような制御が行われる．また，化学プラントなどの工場システムでは，流量センサの入力に基づいて，ポンプを制御し，流量制御を行うようなシステムが構成される．本研究においては，以後，このような構成においてネットワークに接続する個々の組込みシステムをノードと称する．前述の通り，組込みシステムはそれぞれの用途で専用に開発されることが多いため，同一ネットワークに接続されるノードも多様になる．

このような構成でシステムを構築する場合，例えばコントローラの機能をリソースの余裕がある機器に配置するなど，機能の配置を柔軟に行える一方で，相互接続性や通信品質の向上と制御のロバスト性確保などネットワーク経由で制御を実現するための課題の解決が必要となる[10].

また，組込みシステムがネットワーク接続されるのに伴い，システムの複雑度が増大し，開発コストが増大する．特に，プロトコルスタック等，他機器との通信機能を中心とするソフトウェアの開発コスト増大が課題となり，これを低減するという観点からも，ハードウェア等の下位レイヤに非依存なアーキテクチャ設計や相互接続性を担保する通信プロトコルの開発が重要となる．

(3) IoT システム

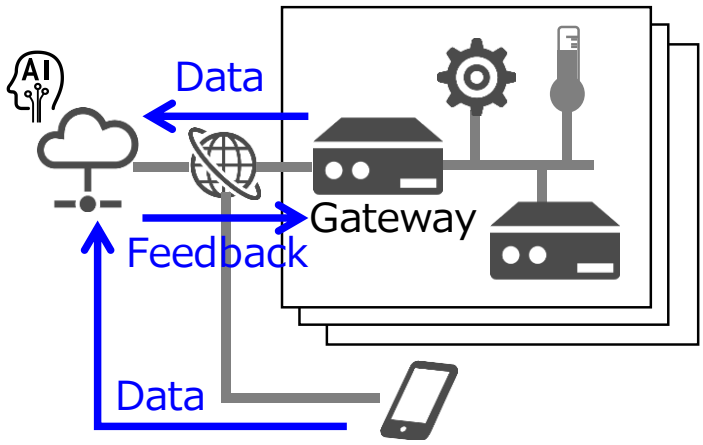


図 1-6 IoT システム

IoT システムは，携帯電話等，直接インターネットに接続されている組込みシステムや，複数のネットワーク接続されたノードからなるシステムがインターネット経由でサーバに接続されて構成されるシステムである（図 1-6）．ネットワーク接続された組込みシステムがインターネットに接続する場合，一般的にはゲートウェイが導入され，ゲートウェイを介してクラウドに接続される．IoT システムでは，これまで単独で動作していたシステムがクラウドに接続され，クラウド-システム間およびクラウドを介したシステム-システム間の連携により生産性の向上など，さらなる価値の向上が見込まれている．このような価値向上は，大量のデータを各システムからクラウドに集約し，そこで処理された結果を個別のシステムにフィードバックすることによって実現される．種々のデータを用いて実世界の状況をサイバー空間に再構成してシミュレーションなどにより改善方式を構築し，実世界にフィードバックして設計から保守までの製品のライフサイクルを改善するこの枠組みは，デジタルツインとも呼称される[11][12][13]．

IoT システムにおいては，インターネット越しに大量のデータのやり取りが発生するため，ネットワーク越しでのデータ収集とそのフィードバックとしての制御が課題となる[14]．また，インターネットに接続することにより外部からの遠隔攻撃などのセキュリティリスクを増大させるという課題が生じる[15]．

1.1.2. 研究の目的

ネットワーク接続されたシステムには図 1-7 に示す 3 つの構成形態がある．すなわち，①ローカルネットワークに接続された複数のノードから構成される形態，② ①がさらにゲートウェイを介してインターネットに接続する形態，③組込みシステムが直接インターネットに接続する形態，である．

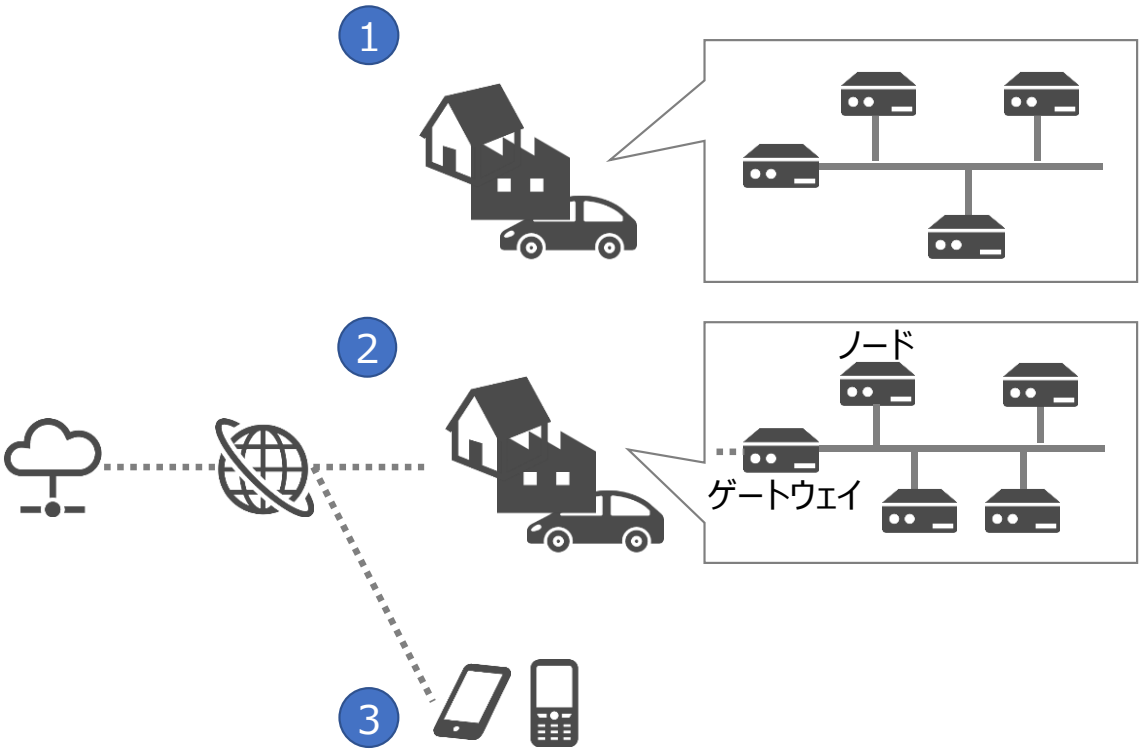


図 1-7 ネットワーク接続された組込みシステムの構成形態

IoT 化したシステムである②および③の最大の特徴は，実システムから収集したデータに基づきサイバー空間で導出した改善策を実システムにフィードバックする枠組みであると言える．このようなフィードバックによる実システムの改善は，これらのフィードバックを反映した新たな製品を設計・開発するか[13]，あるいは運用されているシステムそのものに適用する[11]ことで実現される．後者のアプローチでは，運用中のシステムの再構築または拡張が可能であることが前提となる．本研究では，このような改善手段や拡張機能の実世界への反映を「システム拡張」と称する．運用中のシステムのシステム拡張には，システムに機器を追加する方法やインターネットを経由したソフトウェアのアップデートにより，機能の追加などを行う方法がある．特に③の形態においては，携帯電話のソフトウェアアップデートが広く普及している[16][17]．

②の形態において，ノードのインターネット上のサーバへの接続形態には図 1-8 に示す

２通りがある．すなわち，図 1-8（a）に示すような，更新対象となる機器が IP アドレスを持ち直接インターネットに接続する形態と，図 1-8（b）に示すような，ゲートウェイを介してシステムがインターネットに接続する構成である．前者の構成では，ゲートウェイは IP ルーティングのみを行い，サーバとノードがアプリケーション層の終端となる．後者の構成では，ゲートウェイがアプリケーション層を一度終端し，プロトコル変換などによりサーバとノードを間接的に接続する．前者の構成機器の例として，ゲーム機やネットワーク機能を持つデジタルテレビ・レコーダなどのデジタル家電があり，これらの機器では，ユーザが選択するアプリケーションの追加やセキュリティリスク対策のためのファームウェアアップデートなどが行われている[18][19]．後者の構成については，ホームネットワークシステムや，今後高い成長率でその数が増大すると予測されている自動車や FA システムのような産業用途のシステムなどがこれに該当する．今後は，図 1-8（b）のような構成のシステムにおける，システムの拡張がますます重要となる．

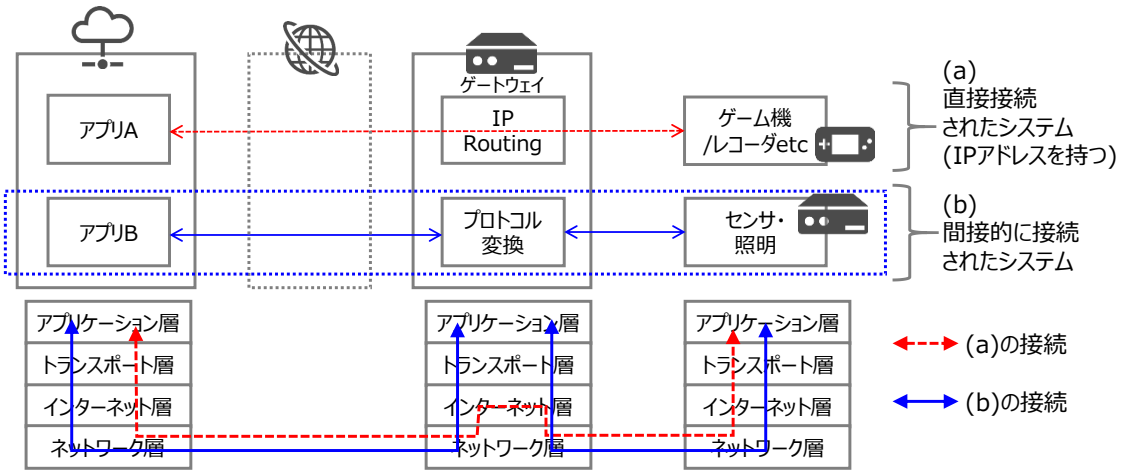


図 1-8 IoT システムのインターネット接続形態

一方，インターネットに接続されない①の形態においても，保守や機能向上のために，放送波を介したソフトウェアのアップデート[20]や，SD カードなどの記録媒体や専用ツールなどを利用した，人手によるアップデートが行われている．

以上述べた環境を踏まえ，本研究では，ネットワーク化された組込みシステムのうち，ノードのインターネット接続に制限がある①および②の図 1-8（b）のシステムに関し，ソフトウェアによるシステム拡張方式についての提案を行う．これにより，今後益々の普及が予想されている IoT システムを含むネットワーク化された組込みシステムのシステム拡張による価値向上の実現に貢献する．

1.2. 本研究のアプローチ

図 1-3 に示すように，これまでは，コンシューマ分野の IoT デバイスが最も普及してきた一方で，今後の成長率については，自動車や産業用途が最も高いと予測されている．

これらのシステムは，パソコンや携帯電話のように単一の組込みシステムがインターネットに接続するのではなく，ローカルネットワークに接続された複数の組込みシステムからなるシステムがゲートウェイを介してインターネットに接続する．ネットワーク接続された組込みシステムのうち，このようなシステムの，パソコンや携帯電話と異なる最大の特徴は構成の多様性である．すなわち，システムを構成するノードやそれらノードを接続するネットワークが多岐にわたる．

一方，IoT システムにおいては，デジタルツインのコンセプトで示されるように，収集した実データを用いたサイバー空間でのシミュレーションなどにより生み出された改善手段や拡張機能を，実世界の組込みシステムに反映，すなわちシステム拡張できることが重要となる． 図 1-8 の①や②（b）に示すシステムにおいて，システムの拡張は，ノードの追加，またはノードの再構成により実現される．ノードの追加や再構成によるシステム拡張を実現するためには，システム拡張を可能または容易にするノードの構成における課題の解決と，システム拡張の実行における課題の解決が必要である．

このような環境において，本研究では，システム拡張を容易にする構成という課題に対して，アプリケーション拡張性を考慮した標準プロトコル用ミドルウェアの設計手法を提案する．また，システム拡張を実行する過程における課題に対して，システム多様性を考慮した遠隔ソフトウェア更新制御機能の設計手法の提案，およびリソース制約を考慮したソフトウェア差分更新方法の提案を行う．（表 1-1）．

表 1-1 本研究の提案

#	項目	個別課題	
(1)	システム拡張を容易にする 構成における課題	A	アプリケーション拡張性を考慮した 標準プロトコル用ミドルウェアの設計手法
(2)	システム拡張を実行する 過程における課題	B	システム多様性を考慮した 遠隔ソフトウェア更新制御機能の設計手法
		C	リソース制約を考慮した ソフトウェア差分更新方法

1.2.1. システム拡張を容易にする構成に関する課題の解決

1.1.2 節に述べた通り，ネットワーク接続されたシステムの拡張においては，システム拡張，すなわちノードの再構成を可能または容易にするノードの構成技術が必要になる．このような構成技術としてはハードウェア[21][22]及び OS レイヤからミドルウェアレイヤまでのソフトウェアの各レイヤについてさまざまな技術が提案されている[23]．例えば，OSGi フレームワーク[24]もその一例である．ゲートウェイをコントローラとしてノードの制御アプリケーションを搭載する集中型のシステムにおいては，ソフトウェアコンポーネントの動的な管理を規定した OSGi フレームワークをゲートウェイに適用することにより，ノードの追加に合わせたアプリケーションの動的な更新や追加などによるシステム拡張が容易となる[25][26][27]．また，このような構成においては，通信機能は共通のミドルウェアが担い，アプリケーションは当該ミドルウェアを共通して利用する構成をとることが多い[28][29]．これにより，アプリケーション毎に通信プロトコルを実装する必要がなくなり，アプリケーションの開発工数が低減でき，システムの拡張性が向上する．このような集中型のシステムの例としてホームネットワークがある．

A. アプリケーション拡張性を考慮した，標準プロトコル用ミドルウェアの設計手法

以上述べたように，ホームネットワークのようなシステムでは， OSGi フレームワーク搭載したゲートウェイに標準プロトコル用ミドルウェアを搭載することで，システム拡張を容易化できる．このため，アプリケーションの拡張性が考慮された標準プロトコル用ミドルウェアを設計・導入することがシステム拡張を容易にする構成を実現する課題となる．

一方，近年ホームネットワークは，家庭のエネルギー消費量を削減するホームエネルギーマネジメントシステム（HEMS）の構成手段として再注目されている．このような中で，国内では HEMS 向けの標準プロトコルとして ECHONET Lite（エコーネットライト）[30]が新たに規定され，2012 年には公知の標準インタフェースとして推奨された．

以上を踏まえ，本研究では，ゲートウェイ上にアプリケーションとミドルウェアを搭載して構成するシステムの代表例としてホームネットワークシステムを取り上げ，新たに規定された ECHONET Lite プロトコル用に，アプリケーションの拡張性を考慮したミドルウェアの設計手法について提案する．本提案では，新規に標準化された ECHONET Lite プロトコルの適用における課題を明確にし，対象分野おける先行技術を踏まえて，課題を解決する．

1.2.2. システム拡張を実行する過程に関する課題の解決

1.1.2 節に述べた通り，ネットワーク接続されたシステムを拡張するに当たっては，システム拡張を実行する過程における課題の解決が必要となる．システム拡張はノードの追加またはノードの再構成に行われるが，ノードの再構成によるシステム拡張では最初に，更新データの生成，ノードへの配信，ノード上での適用，という過程が実行される．近年，さまざまな分野でソフトウェア再構成によりシステム全体を再構成可能にする Software defined 化が進展しており，組込みシステムのソフトウェアの遠隔更新についても，前述のとおり携帯電話やゲーム機などのコンシューマエレクトロニクス機器で広く運用されている．また，センサネットワークシステムや自動車システム向けにも，ネットワークを介して効率的にソフトウェアの配信を行う技術が提案されている[31] [32]．これらの提案では，送信制御や差分更新により，ネットワーク帯域の有効利用や消費電力の削減が行われている．

一方で，本研究の対象とするネットワーク接続されたシステムは，メーカや機能，特性の異なる多様なノードが接続されて構成される．メーカや機能が異なることで，ソフトウェアの更新手順が異なる場合がある．また，用途や機能・特性が異なることから，CPU の動作周波数や ROM，RAM などのノードのハードウェアリソースも，大小さまざまである．このようなシステムにおける更新方法として，ノード毎の更新制御機能を OSGi バンドルとしてゲートウェイに相当する機器に配信し，当該機器から更新を制御することで，ノードの多様性に対応する技術もある[33]．本研究では，このような構成の具体例として，自動車システムを対象とする．

B. システム多様性を考慮した，遠隔ソフトウェア更新制御機能の設計手法

図 1-8 の（b）に示すシステムにおいて，ノードの管理はゲートウェイが提供する基本サービスの一つであり，ノードのソフトウェアの更新はその一部である．例えば，[31]や[32]では，ネットワークの帯域有効利用や消費電力の削減のために，送信制御方式や差分更新方式が提案されてきた．しかし，これらの提案において，センサノードの多様性については言及されていない．一方，前述のとおり，本研究で対象とするシステムはさまざまなノードから構成され，多様である．このようなシステムにおいては，ノードの多様性に対応可能な更新制御の実現が課題となる．

現在の自動車システムは，多数の，用途の異なる Electronic Control Unit (ECU)で構成されて機能を実現する分散型システムであり[34]，構成要素となる ECU は，機能ごとに異なる特徴がある．例えば，自動運転機能を担う ECU と，エンジンやブレーキ制御を行う

ECU，ライト点灯制御を担う ECU では，動作周波数や RAM，ROM などのリソースが大きく異なるほか，信頼性への要求などが異なる．このことから，ソフトウェアの更新における制御方法もそれぞれの ECU 毎に異なる場合がある．近年，自動車の IoT 端末化の進展に伴い，自動車分野においても，制御用 ECU の遠隔でのソフトウェア更新の導入について，サイバーセキュリティと合わせて国連での法規化[35]や国土交通省の自動運転等先進技術に係る制度整備小委員会[36]での国内でのルール作りなどの検討が進められており．次世代の交通システム実現に向けた重要な要素となっている．

以上を踏まえ，本研究では，このような構成の代表例として，自動車システムのゲートウェイ機能に搭載する遠隔ソフトウェア更新制御機能の設計手法について提案する．本提案では，自動車システムに遠隔ソフトウェア更新を適用する際の課題を明確にし，これを解決するアーキテクチャを提案する．

C. リソース制約を考慮した，ソフトウェア差分更新方法

携帯電話のソフトウェアを遠隔で更新する際の技術として，通信データを削減する差分更新技術や部分的な更新により書き換え領域を削減して更新時間を短縮する技術が提案されている[16].

一方，本研究の対象とするシステムでは，ゲートウェイ以外のノードは直接インターネットに接続されず，携帯電話等と比較して機器のリソースの制限が厳しい．例えば，プログラムやデータを格納するための ROM や RAM などのメモリは限られ，CPU の動作周波数も低く抑えられている．ノードを接続するネットワークに関しても十分な帯域がない場合も考えられる．さらに，ノードの用途によっては停止時間に制限があり，無停止または短い停止時間での更新が必要となる．以上から，対象とするシステムでは，限られたネットワーク帯域及びメモリの少ない環境でも，短時間に更新を行うことが課題となる．

本研究では，このような構成の代表例として，自動車システムの ECU のソフトウェアを差分更新する手法について提案する．本提案では，十分なメモリを持たない機器に差分更新を適用する際の課題を明確にし，これを解決する．

本研究では，以上の提案により，拡張可能な組込みシステムの実現を目指す．

1.3. 本論文の構成

本節では，本論文の構成について示す．まず 2 章で，本研究の対象であるネットワーク接続された組込みシステムについて，基本的な構成と，システム拡張のための構成及び過程の

課題解決に関する関連研究を述べる．そのうえで，具体例として論じるシステムの概要とそのシステム拡張における課題を述べ，本研究の位置づけを明確にする．

3 章では，システム拡張を容易にする構成に関する課題の解決のため，アプリケーションの拡張性を考慮した，標準プロトコル用ミドルウェアの設計手法について論じる．具体的には，ホームネットワークシステムを対象に，サービスゲートウェイに搭載する ECHONET Lite ミドルウェアを OSGi フレームワークのバンドルとして設計し，当該プロトコルの運用課題を解決する方法を提案する．

4 章では，システム拡張を実行する過程に関する課題の解決のため，システム多様性を考慮した遠隔ソフトウェア更新制御機能の設計手法について論じる．具体的には，自動車システムを対象に，遠隔でのソフトウェア更新に必要となる更新制御機能を車載ゲートウェイ上の機能として設計し，自動車システムへの遠隔ソフトウェア更新適用の課題を解決する方法を提案する．

5 章では，システム拡張を実行する過程に関する課題の解決のため，ノードのリソース制約を考慮した差分更新方式について論じる．具体的には，自動車システムを構成する ECU を対象に，ネットワーク帯域と，ECU のメモリリソースの制約上でソフトウェア更新時間を短縮する差分更新技術を提案する．

図 1-9 に 3 章，4 章，5 章の位置づけを示す．

最後に 6 章において，本研究を総括するとともに，今後の展望を述べる．

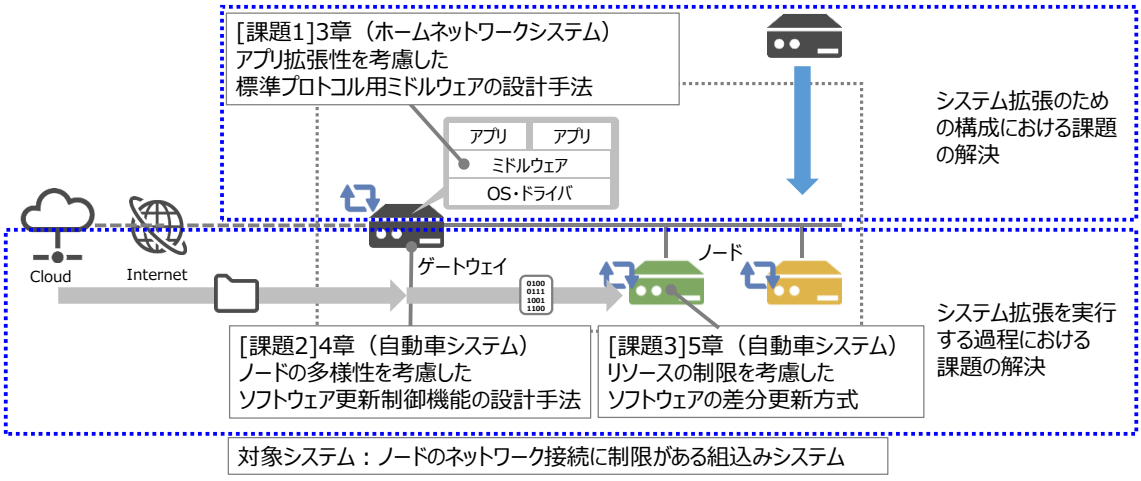


図 1-9 本論文の構成

第2章 対象システムの基本構成と関連研究

本章では，最初にネットワーク接続された組込みシステムの基本構成と，ネットワーク化されたシステムで必須となる相互接続技術に関して述べる．次に，システム拡張に関連する技術について述べ，その具体例として本研究の対象とするシステムであるホームネットワークシステムと自動車システムの構成について述べる．

本研究で対象とするシステムは，図 1-7 の①に示した複数のノードがローカルネットワークに接続して構成されるシステムと，②のシステムのうち，図 1-8（b）に示したノードがゲートウェイを介して間接的にインターネットに接続されるシステムとする．

2.1. 基本構成と相互接続性

本節では，ネットワーク接続された組込みシステムの基本構成と，ネットワーク化されたシステムに必須となるノードの相互接続に関する既存技術について述べる．

2.1.1. ネットワーク接続された組込みシステムの基本構成

ネットワーク接続された組込みシステムがノード間で連携して動作する際の基本構成として，集中型と分散型の 2 通りの構成がある．図 2-1 に基本構成の概要を示す．

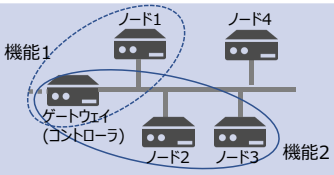
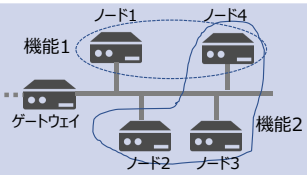
	(a)集中型システム	(b)分散型システム
機能配置		
例	ホームネットワーク，ビルシステム	自動車システム，工場システム

図 2-1 ネットワーク接続された組込みシステムの基本構成

集中型のシステムでは，ゲートウェイなどに搭載されたコントローラが，複数のノードを制御して機能を実現する．例えば，図 2-1（a）に示すように機能 1 はゲートウェイ（コントローラ）とノード 1 で実現し，機能 2 はゲートウェイとノード 2，ノード 3 で実現する．集中型のシステムの例としては，ホームネットワークシステムやビルシステムがある．

分散型のシステムでは，ノードがそれぞれ連携して機能を構成する．例えば，図 2-1 (b) に示すように機能 1 はノード 1 とノード 4 で実現し，機能 2 はノード 2 とノード 3，ノード 4 で実現する．分散型のシステムの例としては，自動車システムや工場システムがある．

システムによっては，集中型と分散型の両方の要素を含む場合もある．

2.1.2. ノードの相互接続技術

ネットワーク接続された機器同士を連携させるためには，通信の標準化による相互接続性の担保が重要となる．機器追加等によりシステム拡張を行う場合においても，互換性のない通信方式の機器・機能を追加・連携させることは一般的に困難なためである．

相互接続性の確保に当たっては，物理層から，各種命令やデータの中身などを規定するアプリケーション層までの規定が必要である．物理層からアプリケーション層まで，いずれもシステムの特性を考慮した技術が標準化され利用されている．例えば，ホームネットワークシステムでは物理層は Ethernet[37]，WiFi[38]を中心に，電力線通信や 2.4GHz 帯無線通信が，ネットワーク層～トランスポート層には TCP,UDP/IP,6LoWPAN が利用される．また，アプリケーション層は UPnP[39]や DLNA[40]，ECHONET Lite などが利用されている．工場システムやビルシステムでは，フィールドバスの物理層には RS-485 や Ethernet などが利用され，ネットワーク層～トランスポート層には TCP,UDP/IP が利用される．また，アプリケーション層には PROFIBUS[41]や Modbus[42]，EtherNET/IP[43]などが利用される．自動車システムでは，車内ネットワークの物理層~データリンク層には CAN[44]や LIN[45]，MOST[46],FlexRay[47],Ethernet[48][49]が利用され，ネットワーク層～トランスポート層には CAN-TP[50]や TCP,UDP/IP などが利用される．また，アプリケーション層は診断通信に UDS[51]などが利用される一方，制御は各自動車メーカ独自のメッセージであることが多い．表 2-1 各分野における標準化された通信プロトコルの例を示す．

表 2-1 標準化された通信プロトコルの例

	ホームネットワーク システム	工場システム ビルシステムなど	自動車システム
アプリケーション層	UPnP,DLNA, ECHONET Lite Zigbee	PROFIBUS ,MODBUS, EtherNET/IP	UDS
トランスポート～ ネットワーク層	TCP,UDP IP,6LoWPAN	TCP,UDP IP	CAN-TP TCP,UDP IP
データリンク～ 物理層	Ethernet,WiFi, 電力線通信 Zigbee,Bluetooth Wi-SUN	RS232C,RS485 CAN Ethernet	CAN,LIN, FlexRay,MOST Ethernet

2.2. ネットワーク接続された組込みシステムの拡張性に関する研究

1.1.2 節で述べたように，組込みシステムの拡張には，異なる製品への適用と，市場にあ

るシステムの拡張の 2 通りがある（図 2-2）.

異なる製品への適用とは，ある製品をベースにして，ハードウェアが変更されたり機能が拡張された他の製品を開発することであり，このような場合も拡張性を考慮したソフトウェアの開発が必要になる．このような他製品への展開のための技術としては，ハードウェアなどの下位層を隠蔽してソフトウェアモジュールの配置の自由度や再利用性を高めるようなアーキテクチャや，他製品への展開を考慮した設計手法がある．アーキテクチャの例としては，標準的なドライバ API の定義や RTOS へのドライバの着脱管理機構導入によってミドルウェアの再利用性を高めた T-Engine アーキテクチャ[52]や，さらにアプリケーション間のインタフェース定義方法や開発手法を規定して ECU プラットフォーム開発のフレームワークを規定する AUTOSAR アーキテクチャ[53]がある．また，設計手法に関しては，共通部品と可変部品を組み合わせて製品を開発していくソフトウェアプロダクトライン[54]や，ソースコードから拡張性に問題がある個所を特定してこれを改善する手法[55]が提案されている．

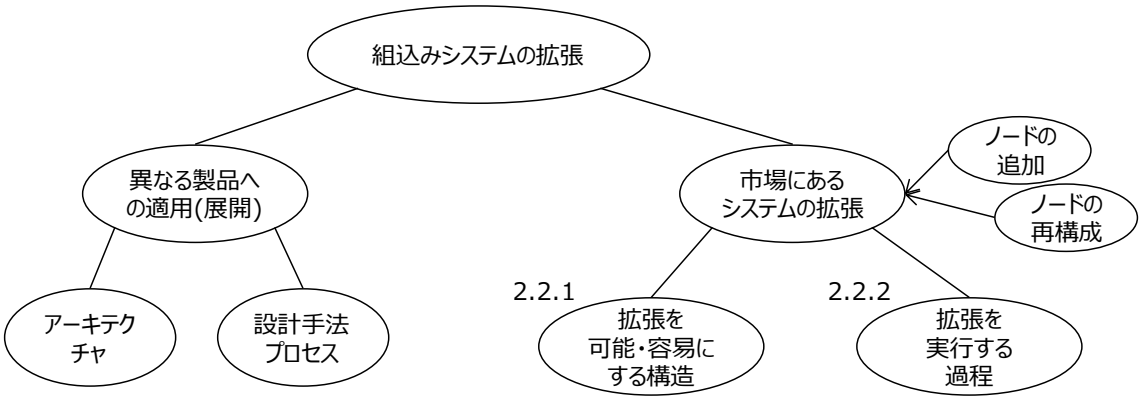


図 2-2 組込みシステムの拡張性に関する技術

一方，市場にあるネットワーク接続された組込みシステムの拡張は，ノードの追加または，ノードの再構成により実現される．ノードの追加の事例としては，ホームネットワークシステムにおいて，新規購入した機器を追加した際に，当該機器もエネルギー管理の対象とするような拡張が行われる場合がある．また，ノードの再構成の事例としては，自動車システムにおいて，ECU のソフトウェアを更新することで，航続距離を延ばしたり，自動運転が可能になるなどのシステム拡張が行われる場合がある．他にも，ノードの追加や再構成に合わせて，システム内のネットワークを追加・再構成するようなケースもある．このような拡張を可能にする技術としては，システムの再構成を可能または容易にする構成方法に関する技術と，拡張を実行する過程に関する技術がある．以下，2.2.1 節において拡張を容易にするノードの構成方法に関する先行研究について，2.2.2 節において拡張を実行する過程に関する先行研究について述べる．

2.2.1. システム拡張を可能・容易にする構成に関する技術

システムの拡張のアプローチの一つに，ノードの再構成がある．本節では，ノードの再構成，特に動的な再構成を可能にする構成方法に関する技術，及びノードの再構成を容易にする構成方法に関する技術について述べる．ノードの再構成の対象は，（1）ハードウェア，（2）ソフトウェアのいずれかまたは両方である．

(1) ハードウェアの構成に関する技術

一般的に，組込みシステムが出荷された後で，例えば，基板に部品を追加するなど出荷されたハードウェアを変更することは困難である．

一方，デジタル回路については，近年，内部構成回路の書き換えによりハードウェアを再構成するシステムが研究，実用化されている[21][22][56][57].このような Reconfigurable System の例としては Field Programable Gate Array（FPGA）や Dynamic Reconfigurable Processor（DRP）がある．再構成可能システムとして，CPU と FPGA を組み合わせたシステムが Xilinx 社[58]等から発売され，利用されている．また，DRP を搭載したデジタルカメラも発売される[59]など FPGA 以外の再構成可能なハードウェアの普及も始まっている．

(2) ソフトウェアの構成に関する技術

ソフトウェアの再構成は一般的にはソフトウェア更新と呼ばれ，様々な分野で広く利用されている．

図 2-3 に示すようにソフトウェア更新はストレージに格納されたバイナリイメージ全体からパラメータまで様々な範囲で再構成が可能である一方で，更新範囲によって柔軟性や必要なコストが異なる[23].

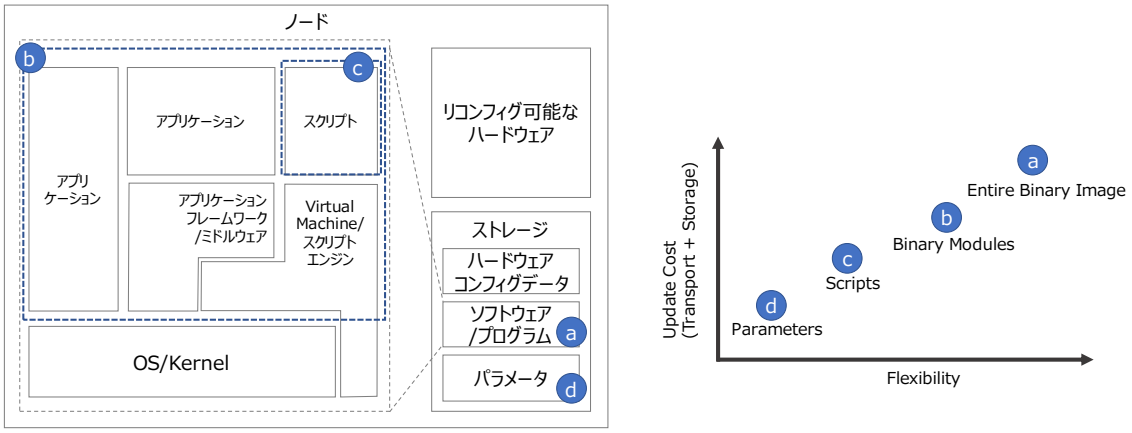


図 2-3 再構成に関するノードの構成とソフトウェアの更新範囲の比較

バイナリイメージ全体を更新する場合，動的な拡張を可能とする構造への考慮は不要であるが，2.2 節の冒頭で述べたような設計手法やアーキテクチャ選定が拡張のし易さに影響する．一方でバイナリモジュールやスクリプトなど，ソフトウェアの一部を更新する場合は，このような部分的な更新や，部分的な更新を動的に実行することを可能とするアーキテクチャでノードを構成することが課題となる．このような部分的なソフトウェア更新を可能にするソフトウェアの構造に関しては，OS，ミドルウェア／アプリケーションフレームワーク，バーチャルマシン／スクリプトエンジンなど様々なレイヤでの取り組みがなされてきた．

OS レイヤでは，リソースの少ないセンサネットワークシステムのセンサノードなどに向けに SOS[60]や Contiki[61]などが提案されている．これらの提案では，プログラムモジュールの位置独立化やメモリ再配置により，ソフトウェアモジュールの動的な追加を可能にしている．また，iTronOS 向けにソフトウェアモジュールの動的な追加を可能にするミドルウェアとして，RLL や DLM が開発されている[62]．

Java[63]に代表されるようなバーチャルマシン（VM）技術も，ソフトウェアの動的な再構成を可能にする．組込みシステム向けには，Android で利用される DalvikVM や Android Runtime（ART）がある[64]．また，低リソースなセンサネットワークノード向けに，Squawk[65]や Mate[66]，VM*[67]などが提案されている．さらに，ノード上でスクリプトを解釈して実行するようなスクリプト技術も VM 技術の一部でありシステム拡張を容易にする．このようなスクリプト技術のうち，組込みシステムでも利用が可能な軽量なものとしては Lua[68]，mruby[69]，konoha[70]などがある．また，よりリソースの制限されたセンサノード向けのスクリプトとして SCript[71]が提案されている．バーチャルマシン技術を用いることにより，アプリケーションやスクリプトなどのモジュール単位でソフトウェアを更新することが可能になる．

さらに，動的な再構成が可能な環境においては，アプリケーションの管理やアプリケーション間の連携も課題となる．このために，前述の OS や VM をベースとして，アプリケーションのライフサイクル管理などを行うフレームワークや，通信プロトコルをサポートするミドルウェアが開発されている．Andorid 向けの Java API フレームワーク[64]や，サービスゲートウェイ向けに開発された前述の OSGi フレームワークが前者に該当する．また，ECHONET で定義された通信ミドルウェアなどが後者に該当する．

OSGi フレームワークは，開発当初，ホームネットワークのサービスゲートウェイ向けの実行基盤として規定されたが，車載機[26][72][73]をはじめとする様々な分野への適用が行われている．図 2-4 に OSGi の階層モデルを示す．

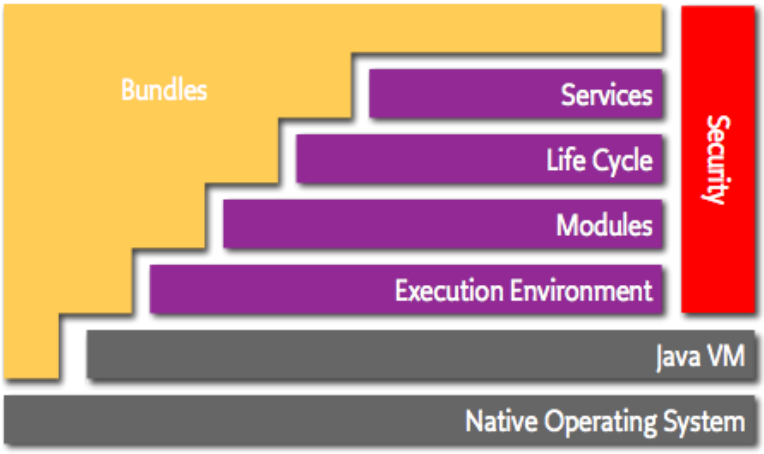


図 2-4　OSGi 階層モデル[24]

OSGi フレームワークは，1 つの JavaVM 上でバンドルと呼ばれるソフトウェアモジュールの管理を行う実行基盤であり，1 つの JavaVM 上でバンドル＝サービスの動的な追加や実行が可能である．すなわち，複数のバンドルが稼働しているときに，特定のバンドルだけを停止・再起動したり，更新したりすることが可能であり，環境や提供するサービスが異なっていても，必要なバンドルの組み合わせで適切なシステムを構築することが可能となる．

さらに，ネットワーク接続されたシステム向けにもアプリケーションの開発を効率化して拡張性を高めることは課題となる．例えば，ノード間の通信プロトコルをアプリケーションごとに実装する必要がある場合などは開発が非効率になり拡張も容易ではなくなる．このため，ネットワーク接続されたシステム向けにも拡張性を考慮した構造が研究され，図 2-5 に示す集中型・分散型のシステムにそれぞれ適用されている．集中型システムの場合，各種サービスの実現はゲートウェイに搭載されたアプリケーションが担い，通信機能は共通のミドルウェアとする構成をとることが多い（図 2-5 (a)）．このような構成においては，ゲートウェイのアプリケーションを更新する，またはゲートウェイに新たなアプリケーションを追加することで，システムが拡張される．分散型システムの場合，各ノードが共通のミドルウェアまたはフレームワーク上にアプリケーションを構築し通信プロトコルなどが隠蔽されたインタフェースで連携し，位置透過性のあるアプリケーション実行環境環境を構成する（図 2-5 (b)）．このような構成においては，アプリケーションの更新や追加に加え，アプリケーションの再配置などによりシステムが拡張される．

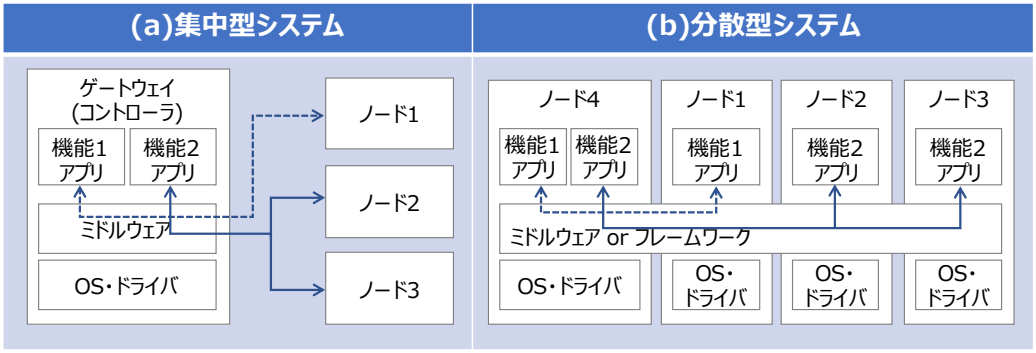


図 2-5 ネットワーク接続された組込みシステムの構成とアーキテクチャ例

集中型システムにおいて拡張性を考慮した構成方法としては，前述の OSGi が広く普及している．OSGi では，モジュールの管理に必要となるバージョン管理，依存関係管理，アクセス制御などの基本機能に加え， UPnP や HTTP など標準的なプロトコル向けのインタフェースもサービスとして規定されている．OSGi を用いたサービスゲートウェイでは，ネットワークへのノードの追加などに伴い，当該ノードの制御を行うバンドルをアプリケーションとして追加する．また前述のとおり，ECHONET では，通信ミドルウェアを規定し，通信ミドルウェアとアプリケーション間のアプリケーションインタフェース（基本 API）を定義して，アプリケーションから共通に利用されるミドルウェア仕様を定義することで，アプリケーション開発の容易化を図っているが，アプリケーションは ECHONET のプロトコルを意識して実装する必要がある．これに対し，OSGi フレームワークをベースに，プロトコルを意識しない，機器を抽象化したインタフェースを定義して多様な機器を統一的に管理・制御するためのホームネットワーク向けのフレームワーク[74]や，ホームネットワーク全体を OS と見立てて，各種機器の制御アプリケーションを PC のアプリケーションと同様に扱う HomeOS[75]が提案されている．これらの構成方法を適用し，共通のミドルウェアを利用する構成をとることで，アプリケーション開発の効率向上が見込まれ，ノードの拡張が容易になる．

分散型システムの構成方法として，例えば前述の AUTOSAR では，Virtual Function Bus（VFB）[76]と呼ばれるアプリケーション間の抽象化インタフェースの概念が導入され，Run Time Environment（RTE）として実装されて同一ノード上またはノード間の通信を隠蔽する．これによりアプリケーションのノードへの配置の自由度を高めている．しかしながら，VFB は，設計時の配置自由度を高めるフレームワークであり，動的なシステム拡張には対応していない．このため，Belaggoun らは AUTOSAR を拡張して動的なソフトウェアコンポーネントの配置を可能にする技術を提案している[77]．Belaggoun らの提案では，提案する動的な再配置機構により，ECU 失陥時の機能再配置を行う例が示されている．他にも企業システム向けに標準化された CORBA（Common Object Request Broker

Architecrure）を組込み向けに軽量化しつつサービス拡張のためのオブジェクトの動的追加機構を追加した Embedded CORBA[78]や，ソフトウェア更新・ノード間連携・動的なサービスの構成により複数のノードを動的に組み合わせて新たなサービスを実行するためのアーキテクチャ[79]が提案されている．

以上述べたように，ソフトウェアの更新によるシステム拡張に当たっては，更新を可能とする OS やバーチャルマシンなどのアーキテクチャと，拡張を容易にするミドルウェアやフレームワークといった技術の導入が重要となる．さらにこれら技術の導入に当たっては，システム構成やシステムで利用される通信プロトコルの特徴を考慮した設計が課題となる．

2.2.2. システム拡張を実行する過程に関する技術

前述の通り，ネットワーク接続された組込みシステムは，ノードの追加やノードの再構成により拡張される．図 2-6 に，システムの拡張手順を示す．ノードの追加の場合は，最初にノードをネットワークに接続する．一方，ノードの再構成の場合は，再構成用のデータ（更新データ）を生成し，ノードに配信し，ノード上で受信した更新データを適用するというステップが必要となる．ノードの追加またはノードの再構成が完了すると，これらのノードまたは再構成された機能を検出することでシステムへの統合が行われた後，他のノードとのデータ送受信や制御が開始され，拡張されたシステムとしての稼働が始まる．これらの過程は，大きく，ノードとしての再構成過程と，システムとしての再構成過程に分類できる．

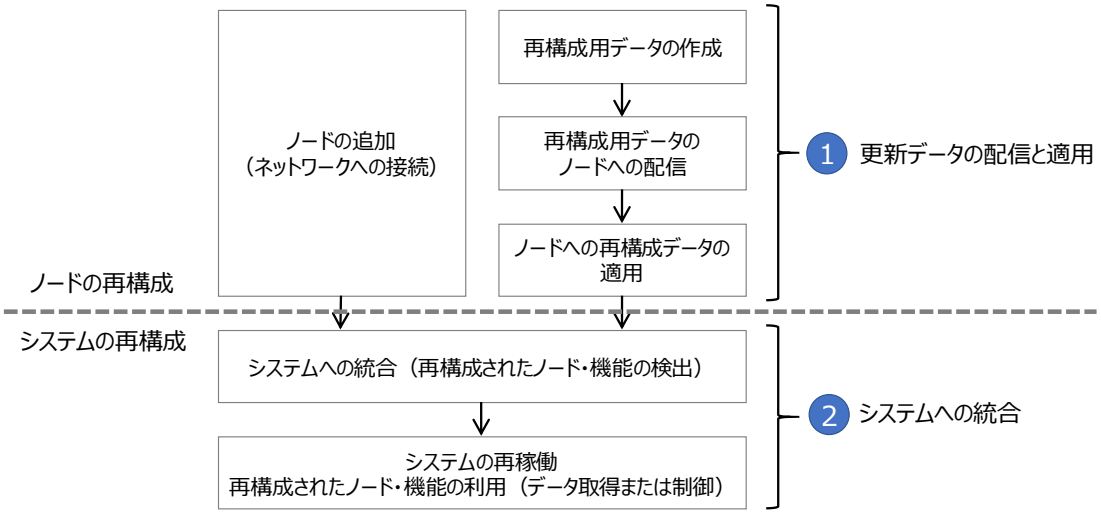


図 2-6 ネットワーク接続された組込みシステムの再構成技術

前述の通り，ノードとしての再構成過程に関する技術として，更新データの配信と適用に関する技術があり，システムとしての再構成過程に関する技術として，システムへの統合に関する技術がある．

(1) 更新データの配信と適用に関する技術

ノードの再構成は，図 2-3 のノード構成図に示すストレージに格納されたハードウェアのコンフィグデータやソフトウェア／プログラム，あるいはパラメータを更新することで実現される．このため，更新するためのデータをどのように生成・配送し，ストレージを書き換えるかが課題となる．特に，ネットワーク接続された機器のソフトウェアを遠隔で更新する技術は，ソフトウェアの大規模化が先行した PC や携帯電話，デジタル放送受信機，ゲーム機などで実用化され，頻繁にアップデートが行われている．この場合の更新データの配信経路としては，インターネットのほかに，携帯電話網や放送波が利用されている．このように携帯電話網等の無線接続を利用してソフトウェア/ファームウェアを端末に配信し，遠隔で更新する技術は OTA (Over the Air software/firmware update：無線によるソフトウェアの更新)と呼称される．また，複数のノードがネットワーク接続されてシステムを構成するようなシステムでは，センサネットワークのセンサノードのソフトウェア更新方式が提案されている．このような，ネットワーク経由でのノードの再構成に当たっては，更新データの作成，ノードへの配信管理･配信，ノードにおける更新の実行が必要である（図 2-7）．このような更新データの配信と適用に当たっては，通信コストの削減・消費電力の低減・配信時間の短縮・メモリなどのリソースの少ない環境での更新・システムあるいはノードのダウンタイムの短縮，ロバスト性が共通の課題となる．

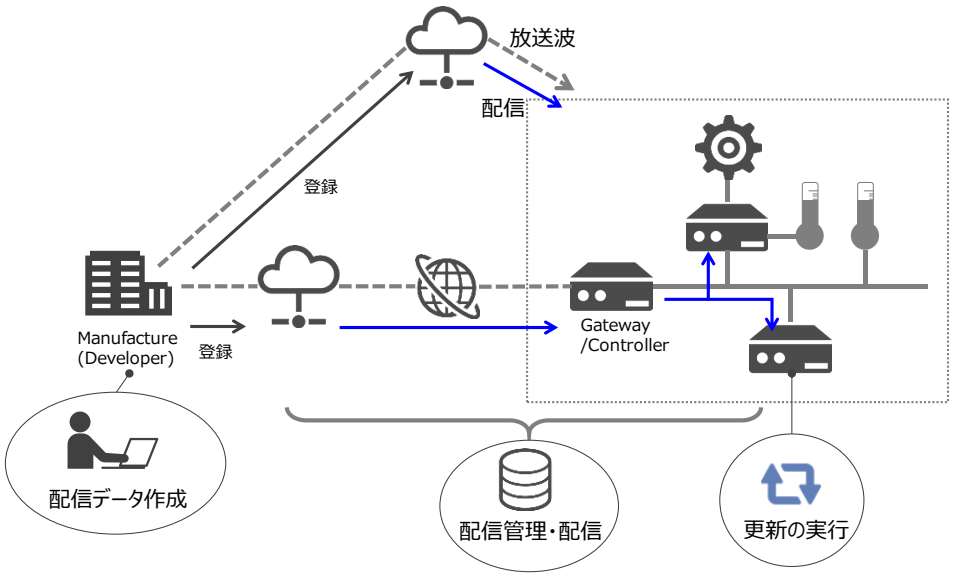


図 2-7 ソフトウェアの遠隔更新システム概要

また，前述の共通課題に加えてそれぞれのフェーズで個別の課題もある．配信データの作成では，ノードの更新のために何をどのような形式で配信データとして構成するかが課題である．配信管理においては配信対象のシステムやノード・更新内容の特定，ノードへの配

信においては End to end でのセキュリティが課題となる．

配信データの形式としては Linux 向けの RPM[80]や Andoriod アプリケーション配布用に APK[64]などのパッケージフォーマットが広く利用されているが，本研究で取り扱うシステム向けに統一的に利用されているフォーマットはいまだ存在しない．

配信管理技術としては OMA DM[81]，Rsync[82]，TR069[83]などが利用されている．OMA DM は Opne Mobile Alliance（OMA）において定められたデバイスの管理を行う標準規格であるが，その中でファームウェアやソフトウェアのアップデート機構として対象特定のための構成同期手順や，ソフトウェアの配信・更新の実行手順を規定している．また Rsync では，サーバとノード間でバイナリレベルでの比較を行い，差分を配信する方式を定義している．OMA では，メモリなどのハードウェアリソース向けのデバイス管理プロトコルとして LwM2M を定義し，その中でも軽量なソフトウェアの更新を定義している[84]．

配信時の課題である通信コストの削減・消費電力の削減・可用性向上を目的とした配信時間の短縮に対応するための共通の手段として，通信データの削減がある．通信データを削減するための差分更新技術も多数提案されている [16][85][86][87][88][89][90]．差分更新では，開発元等で新旧プログラムから差分ファイルを生成して更新対象に転送し，更新対象上の旧プログラムと差分ファイルから新プログラムを復元する(図 2-8)．このようにプログラム全体ではなく差分のみを送信することでネットワーク上を伝送するデータ量を削減できる．

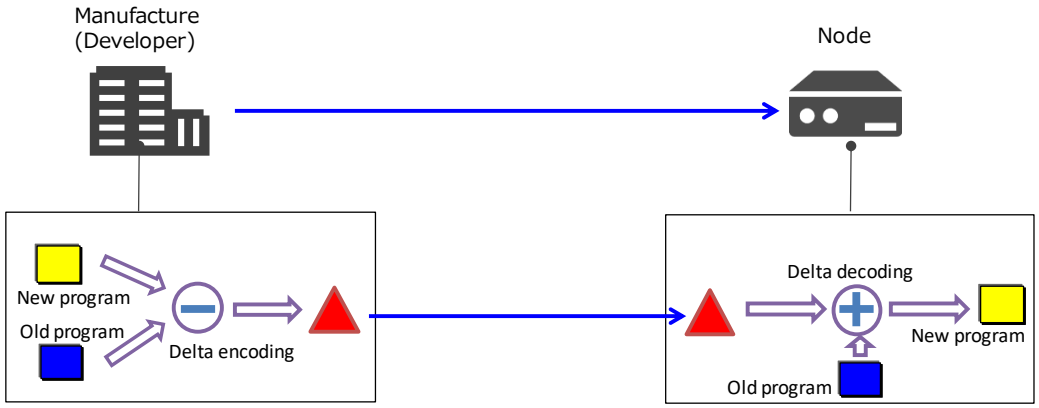


図 2-8 差分更新の概要

図 2-9 に前述の差分圧縮アルゴリズム[87][88][89][90]を車載ソフトに適用して評価した場合の，圧縮効果を示す．本評価によると，bsdiff の圧縮効果が最も高い．

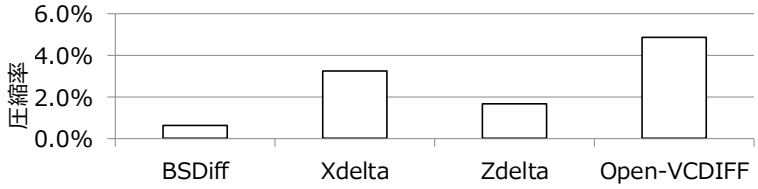


図 2-9 差分圧縮の効果

ワイヤレスセンサネットワークの分野では，配信時に加えノードへの更新適用時の課題にも対応したネットワーク経由でのソフトウェア更新方式が提案されている[91]．宮丸らは，消費電力や配送速度の改善のために，配送のパイプラインのセグメント分割数をローカライズして最適化する方式を提案している[31]．また，Tsiftes らは，複数の圧縮アルゴリズムを比較し，ネットワーク伝送量とノード上での処理時間のトレードオフを行い，GZIP が最も消費電力を削減できることを示している[92]．さらに，Stolikj らは，複数の圧縮アルゴリズムと差分圧縮アルゴリズムについて同様のトレードオフを行い，bsdiff と LZ77 の組み合わせが最も好適であることを示している[93][94]．他にも，中西らは自動車向けに更新途中で給電が停止した場合も復旧後に更新を継続できる，ロバスト性を考慮した差分更新方式を提案している[32]．

また，配信時のセキュリティを考慮したセンサネットワーク向け[95]や自動車システム向け[96][97]の配信方式が提案されている．

以上，ソフトウェア更新の先行研究について述べたが，ハードウェアのコンフィグデータをソフトウェアの更新データと同様に扱えば，ソフトウェアの遠隔更新の枠組みは，再構成可能なハードウェアを遠隔更新する場合にも適用できる．例えば，永田ら[98]は FPGA の遠隔再構成システムを提案している．

前述のとおり，従来，遠隔でのソフトウェア更新が適用されてきた分野は携帯電話やスマートフォンなど，直接インターネットに接続する組込みシステムか，同種のセンサノードから構成されるセンサネットワークが想定されていた．このため，システムが多様なノードから成る構成のソフトウェア更新方法については十分に検討されておらず，その更新制御が課題となる．

また，低消費電力化やネットワーク帯域の有効活用のために差分更新を適用するに当たっては，ROM，RAM などのノードのハードウェアリソースを考慮した方式の適用が課題となる．

(2) システムへの統合に関する技術

あらたに接続されたノードや，あらたに構成された機能をシステムに組み込むためには，

システムが接続されたノードや機能を認識し，連携できるようにする必要がある．このような管理を，ノード追加を行った人間が都度，他のノードの設定変更等によりその対応を行うのは困難である．

このため，追加されたノードをシステムに統合するための仕組みとして，ノードや機能の検出の仕組みが必要になる．このような検出の仕組みとして前述の UPnP では，Simple Service Discovery Protocol（SSDP）[99]が利用されており，Apple 社が普及させた Bonjour[100]では multicast DNS（m-DNS）と DNS service discovery（DNS-DS）が組み合わせて用いられる[101]．また，ECHONET Lite などでも独自のノード検出方法が定義されている[30]．他にも，アドホックネットワーク向けの Konark[102]やリソース制約のあるセンサネットワーク向けに Bonjour Contiki[103]が提案されている．自動車分野では現在，AUTOSAR において，サービス検出・利用の仕組みとして SOME/IP[104]が標準化されている．これらの技術では，検出対象をある機能を提供する「サービス」ととらえ，サービスを検出するための枠組みとして定義されている．このようなサービス検出の仕組みを用いることで，利用者はマニュアルで機器の登録作業などを行う必要なく，ネットワークに組み込まれた機器を利用できるようになる．このようなサービスの特定と識別に当たっては，ノードやサービスをトレースするための識別方法も重要となる．

2.3. ホームネットワークシステムとその関連技術

2.3.1. ホームネットワークシステムの基本構成と動向

本研究で取り扱うホームネットワークシステムの基本構成例を図 2-10 に示す．本構成では照明やエアコンをはじめとする白物家電やテレビやレコーダなどの AV 機器，各種センサやスマートメータなどホームネットワークを構成する機器はサービスゲートウェイと接続されることを想定する．サービスゲートウェイは，インターネットなどの外部ネットワークと接続される．

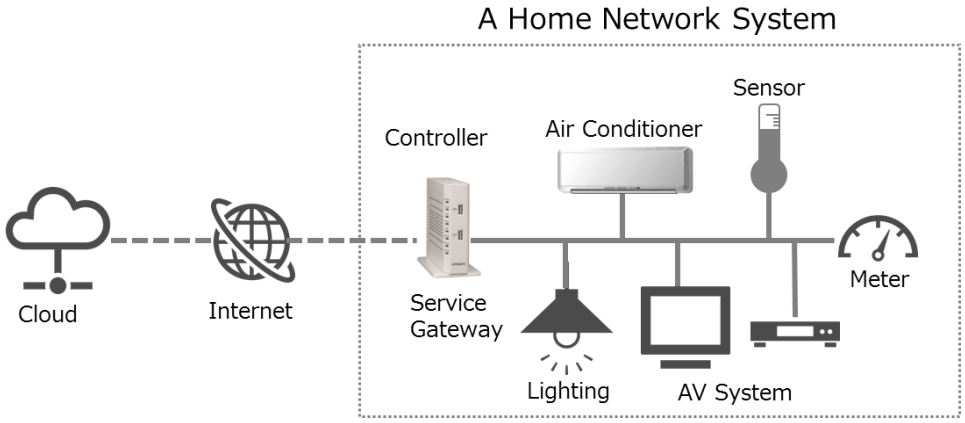


図 2-10 ホームネットワークシステムの基本構成例

ホームネットワークシステムについては，1970 年代以降さまざまな取り組みがなされてきた．例えば，ユーザの利便性を向上するホームオートメーション（HA）機能や，防犯などのセキュリティサービスなどが専用の端末を利用して提供されている．このような状況で，近年，ホームネットワークを活用して家庭のエネルギーマネジメントを行う，ホームエネルギーマネジメントシステム（HEMS）が注目されている．さらに，スマートスピーカなどの導入により，ホームネットワークシステムは，ホームエネルギーマネジメントを中心にセキュリティ（見守り）やヘルスケア，買い物など複数のサービスを提供するプラットフォームへと発展している．

HEMS において，ネットワークで接続される家電機器や住宅設備は家庭ごとに異なっており，機器接続のプロトコル，伝送メディアも多様である．このため，共通の手段で情報取得や制御が行えるような標準技術の重要性が改めて認識されている．

一方，これまで専用の端末上に実装されていた HA やセキュリティ，HEMS といった複数の機能を一つの端末（サービスゲートウェイ，以下サービスゲートウェイ）上に搭載するため，さまざまなサービスやデバイスに対応できるホーム ICT（Information and Communications Technology）実行基盤として OSGi が注目されている[105].

図 2-11 に，サービスゲートウェイに OSGi を利用したシステムの構成例を示す．ここでは，サービスゲートウェイは家庭内の各機器に制御指示を行うコントローラとなる．

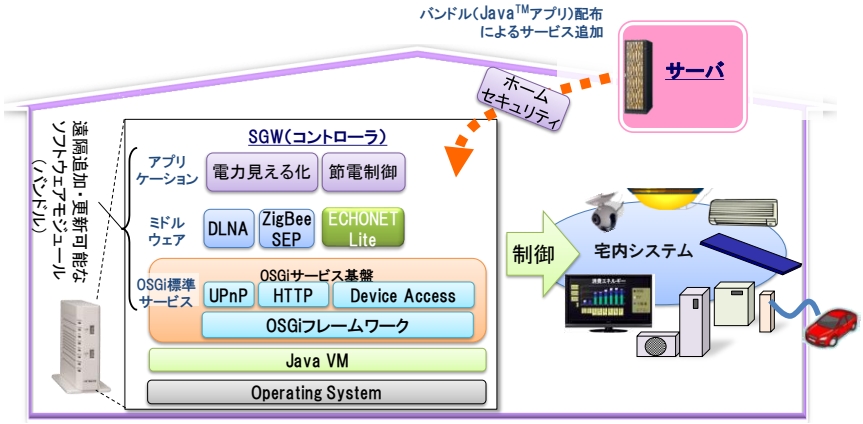


図 2-11 OSGi を利用したホームネットワークシステム概要

このような環境において，経済産業省は 2011 年に ECHONET Lite を HEMS の標準プロトコルとして推奨した[106]．さらに，ECHONET Lite は 2012 年には，日本国内でのスマートメータと HEMS を繋ぐ標準プロトコルとして認定された．そのため，国内においてホームネットワークシステムを構築するためには，ECHONET Lite への対応が必要となっている．

2.3.2. ECHONET Lite

ECHONET Lite は，エコーネットコンソーシアム[107]が策定した通信プロトコルであり，センサ類，白物家電，設備系機器など省リソースの機器を相互に接続し，エネルギーマネジメントなどのサービスを実現するための標準規格である．本規格は，ISO 規格および IEC 規格として国際標準化もされている[108][109].

図 2-12 に示すとおり，ECHONET Lite で規定している範囲は，OSI 参照モデルにおける第 5 層以上となっており，下位の通信仕様は規定しない．

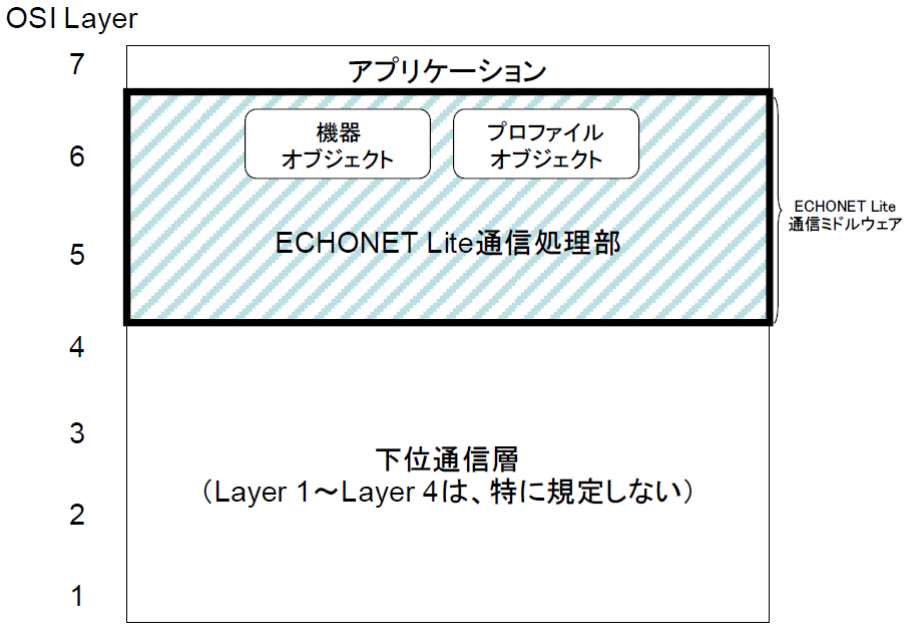


図 2-12　ECHONET Lite の規定範囲[30]

ECHONET Lite 通信プロトコルは，ECHONET Lite フレーム(または，ECHONET Lite 電文)と呼ばれるメッセージをやり取りする通信規格である．本規格では機器を抽象化した ECHONET 機器オブジェクトと呼ばれるオブジェクトが定義され，当該オブジェクトに対するコマンドをやり取りすることで機器のデータ読み出しや操作を行う．ECHONET Lite 機器は，オブジェクト間のコマンドのやり取りを個別通信または一斉同報通信を用いて ECHONET Lite フレームを送受信することで実現する． ECHONET Lite フレームは，ヘッダ(EHD)，トランザクション ID(TID)と，送信元オブジェクト(SEOJ)，送信先オブジェクト(DEOJ)，アクセスルールを指定するサービス(ESV)，アクセス先プロパティ(EPC)，アクセス先プロパティ値(EDT)などから構成される． ESV には，GET(状態取得)，SET(設定・操作)，ANNOUNCE(通知)などがあり，オブジェクトへの操作内容が規定される．

エコーネットコンソーシアムは，ECHONET Lite 採用製品の出荷が 1,000 万台を超えたと公表している[110].

2.3.3. ホームネットワークのノード追加によるシステム拡張における課題

(1) ECHONET Lite を適用したホームネットワークシステムの拡張の容易化

2.2.1 節に述べた通り，システム拡張に当たっては，更新を可能とする OS やバーチャルマシンなどのアーキテクチャと，拡張を容易にするミドルウェアやフレームワークといった技術の導入が重要となる．さらにこれら技術の導入に当たっては，システム構成やシステ

ムで利用される通信プロトコルの特徴を考慮した設計が課題となる．

一方で，前述のとおり，国内では HEMS の標準プロトコルとして ECHONET Lite が推奨されたことから，ECHONET Lite 向けのミドルウェアの開発が進んでいるが，搭載するゲートウェイのリソースなどを含めて拡張性を考慮した実装方法に関する提案は行われていない．具体的には，ECHONET Lite 規格に先立ち標準化された ECHONET 規格では，ソフトウェア実装のための標準 API として，アプリケーションがミドルウェアを利用するための「基本 API 仕様」及びミドルウェアと下位通信層の間の「共通下位通信インタフェース仕様」が定義されており，本仕様を参照した ECHONET 通信ミドルウェアバンドルが OSGi 上で開発されている[111]．また，OSGi 上で ECHONET バンドルを構成し PUCC（P2P Universal Computing Consortium）プロトコルと接続して ECHONET 機器の制御を行うシステムも提案されている[112]．これらの研究では，ECHONET を OSGi に適用する方法について検討されている．しかし，これらの研究はあくまで ECHONET を対象としており，ECHONET と ECHONET Lite の相違点については考慮されていない．例えば，ECHONET では定義されている下位通信層が ECHONET Lite では定義されていないが，その対応方法は検討されていない．さらに，ECHONET Lite 規格を Java 言語で実装したミドルウェアソフトとして，商用ベースの製品[113]やオープンソースで公開されている実装[114]がある．サービスゲートウェイ上でソフトウェアを実装する場合，リソースの制限やサービス停止時間といった課題があるが，これらの実装では，それらについては十分に考慮されていない．

そこで，本研究では，これらの先行研究を踏まえ，OSGi 上の ECHONET Lite に準拠したコントローラ用のミドルウェアとして，ECHONET Lite ミドルウェアバンドルの検討及び評価を行う．

2.4. 自動車システム

2.4.1. 自動車システムの基本構成と動向

図 2-13 に自動車システムの構成を示す．図中の ECU の色が異なるのは，ハードウェアリソースなどの特徴の異なる ECU であることを示す．

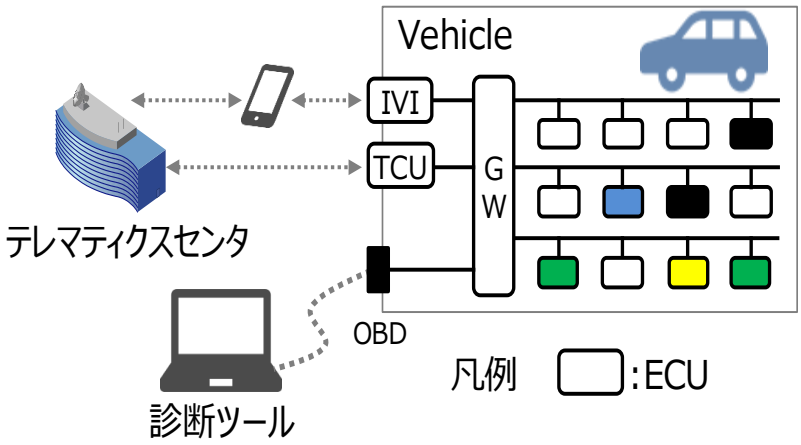


図 2-13 自動車システムの構成例

これまでテレビや携帯電話などのコンシューマエレクトロニクス製品と異なり，自動車システムは 1 つ 1 つが携帯端末に相当するような多数のデバイス（ECU)で構成され，これらが連携して機能を実現する複雑なシステムである．自動車 1 台当たりの ECU の搭載数は 2025 年には平均 30.4 個になると予想されている[115]．

従来の ECU に加えて，カーナビやオーディオ等の機能を備える In-vehicle infotainment (IVI)や車両外部との通信機能を備える Telematics Control Unit (TCU)が搭載され，テレマティクスセンタと通信することでさまざまなサービスが提供されている．

また，車両には整備士が診断ツールを接続して故障診断などを行うための On-board diagnostic (OBD)コネクタが備えられ，現在，ECU のソフトウェア更新は本コネクタ経由で診断ツールから実行される．

近年，これら車両外部とのインタフェースからのサイバー攻撃の事例[116][117]が報告されており，その対策として IVI，TCU や外部接続機器および OBD のような外部インタフェースとの間で Firewall として機能するセキュリティゲートウェイの搭載が進みつつある．

現在，自動車システムは，複数の ECU が連携してエンジン制御や先進安全支援などの機能を実現し，さらにこれらの機能が連携動作することでシステム全体が制御される分散アーキテクチャとして構成されている．一方で，自動車システムは，図 2-14 に示すように各機能を Central ECU が制御する集中アーキテクチャとして構成されるように発展する可能

性が提案されている[34]．ここでは，現在各 ECU に搭載された機能がドメインごとの Central ECU，ドメインをまたぐ Central ECU と段階的に集約され，最終的には Vehicle Computer に集約される．このようなアーキテクチャを支える車載ネットワークとして，従来主流であった CAN と比較して広帯域である Ethernet の導入が始まっている[118][119].

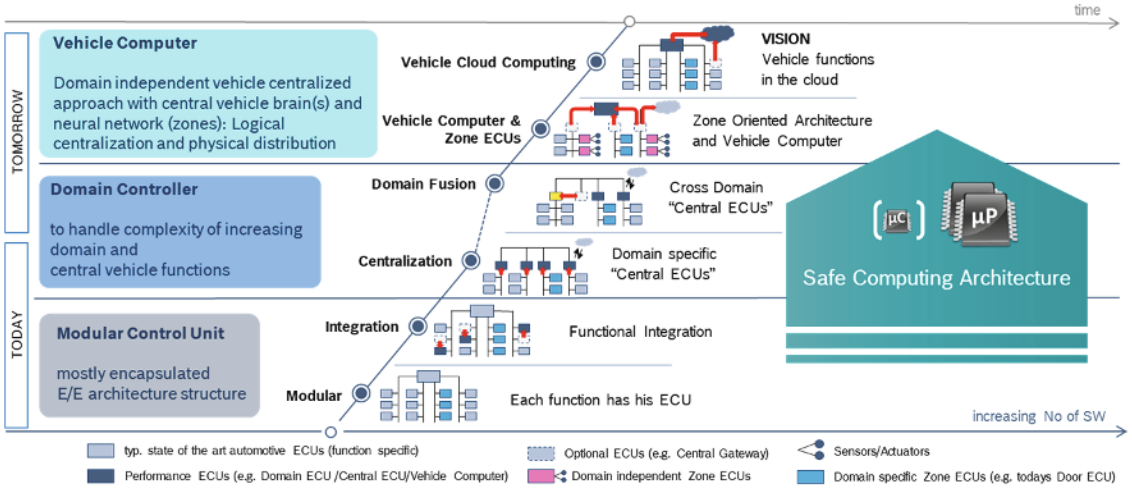


図 2-14 自動車システムの構成の進展[34]

また，制御の複雑化に伴い，図 2-15 に例を示す通り，ECU に用いられるマイコンについても動作周波数の向上やメモリの拡張，マルチコア化など，高機能化が進展している[120].

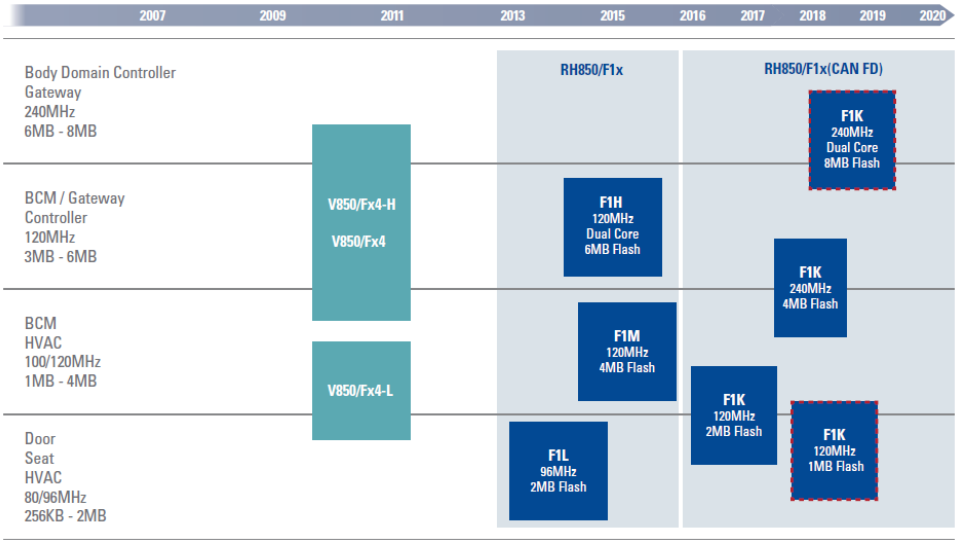


図 2-15 車載 ECU 用マイコンのロードマップ例[120]

さらに，高い演算性能が要求される自動運転機能向けに，GPU や FPGA を組み合わせた SoC の車載適用の検討が進んでいる[121][122].

前述のような自動車システムの高機能化に伴い，これを制御するソフトウェアの規模も増大の一途をたどっている．このため，ソフトウェア更新の所要時間も増大し，更新時間の

短縮が課題となっている[123].

このような中，テスラモーターズ社は，2015 年 8 月には販売済みの車両に対して，ソフトウェア更新によって有料で自動運転機能を追加するという，新たなビジネスモデルを試行している[124].

現在，自動車が販売された後に車載 ECU のプログラム更新を行う場合，顧客（自動車所有者)がディーラに車両を持ち込み，ディーラの整備士が専用ツールを使ってマニュアル作業で 1 台 1 台 ECU 上のプログラムを書き換えることが一般的である．しかし，この方法は，車両の持ち込み等時間がかかることでユーザの利便性が悪い．この問題を解決するため，車載 ECU に対して遠隔ソフトウェア更新を適用し，プログラム更新時のユーザ利便性を向上することが検討され始めている[125].

前述のとおり，デジタルテレビや携帯電話分野では OTA は一般化され，多量の端末のソフトウェアが効率的に更新されている．OTA システムを自動車に適用する場合，新プログラムを OTA センタに登録し，OTA センタは車両に修正した新プログラムを配信する．車両では，新プログラムを受信し，これを更新対象 ECU に適用する(図 2-16).

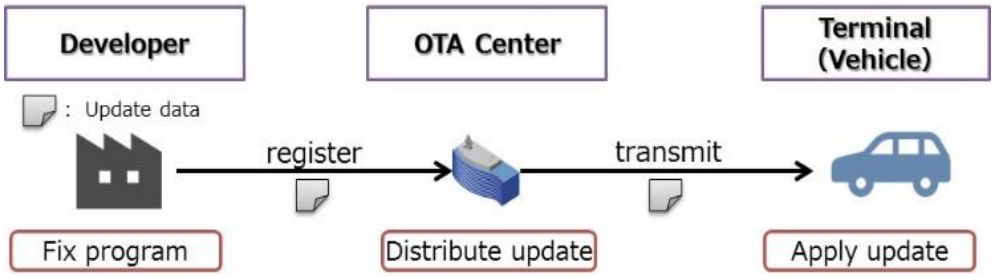


図 2-16 OTA システムの概要

近年，車載機(カーナビ)においても，遠隔での地図やソフトウェアの更新が導入されている．前述の，テスラモーターズ社の取り組みも，OTA を車両に適用することで実現されている．

2.4.2. 故障診断向けの ISO 標準

整備士が，自動車システムの診断や，ECU のソフトウェアを更新する際に用いる専用ツールやツールと ECU の通信インタフェース向けの規格として UDS (Unified Diagnostic Service) (ISO14229)[51]，Open Diagnostic data eXchange (ODX) (ISO22901) [126]，Open Test sequence eXchange (OTX) (ISO13209) [127][128]が標準化されている．

UDS は診断装置と ECU 間の通信規格を定めている．具体的には，CAN 等で接続した ECU との間で送受信するデータを定義するものであり，診断処理 (サービスと呼ばれる)毎にバイナリフォーマットが規定されている．

ODX では前述の UDS で規定された診断サービスで利用するデータの定義を行う記述方式が規定されている．具体的には，ECU に書き込むデータについて，書き込み先のアドレスやサイズ，データ形式などが記述される．他にも，ECU から情報を取得する処理やリセットを行う処理など ECU のソフトウェアを更新する処理で扱うデータの内容が含まれる．これらは XML で記述される．

OTX は, 診断サービスの処理シーケンスを記述するための標準技術である. 具体的には, 前述の UDS で規定され，ODX でデータが記述された診断サービスを，どの順番で実行するか，ECU と通信した結果によって条件分岐する，というような処理の流れを XML で記述する．

図 2-17 に，これら ISO 標準技術の概要を示す．

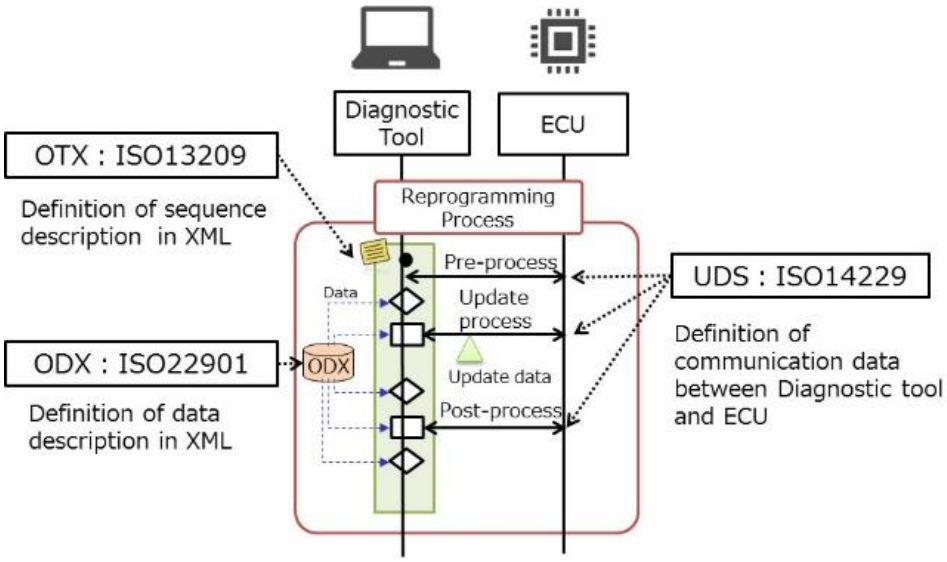


図 2-17 ISO 標準の概要.

このような，標準化された記法で手順を記述することにより自動車メーカ同士や自動車メーカと部品メーカ間で更新制御手続きの共有が容易になる．

2.4.3. 自動車のノード再構成によるシステム拡張における課題

(1) 自動車システムのソフトウェア更新制御における課題

2.2.2 節では，ネットワーク接続された組込みシステムのノードを再構成してシステム拡張を行う場合に，多様性のあるノードから構成されるシステムの更新制御が課題であることを述べた．

一方で，2.4.1 節で述べた通り，自動車の ECU のソフトウェア更新への OTA 適用が検討されている．OTA によるソフトウェアの更新制御に関して，Boer らは ECU ごとの更新手順の相違への対応という課題に対し，Head Unit に OSGi フレームワークを搭載して ECU 毎のシーケンス制御ソフトをバンドル（Java のアプリケーションプログラム）として配信する方式を提案している[33]．しかしながら，近年重要となっているサイバーセキュリティ対応の自動車システム構成を想定した機能配置や提案方式の運用容易性については考慮されていない．

また，Ryu らは，テレマティクスユニットに OMA DM クライアントを搭載し，複数の ECU で構成される自動車の構成管理や，ソフトウェアのアップデートを行うアーキテクチャを提案している[129]．しかしながら，ソフトウェア更新に関して，具体的な振る舞いや OMA DM の規定範囲外である更新のためのパッケージの内容については考慮されていない．

他にも，Zhang らは，携帯電話を経由して ECU ソフトウェアの更新パッケージをダウンロードし，ECU を更新する Onboard platform を用いた OTA ソフト更新システムを提案している[130]．しかしながら，車載システムにおける更新制御機能の要件についての整理やそれに基づく構成の提案はなされていない．

本研究では，前述の研究を踏まえ，自動車システムへの OTA 適用を実現する更新制御方式についての機能要件を整理し，これを実現するシステム構成を提案する．特に，自動車システム特有といえる前述の多様性に着目し，ECU ごとの更新手順の相違に加えて，車種などの多様性への対応も可能な制御方式を提案する．

(2) 車載 ECU のソフトウェア更新における課題

OTA では，通常一般顧客（車両所有者）が更新を行うため，車両を使用できない時間をできるだけ短くすることが要求される．また，停車中も長時間電力供給が可能な電気自動車と異なり，ガソリン車では更新時間の短縮はより重要となる．すなわち，エンジン動作中に更新を行う場合は，更新時間の長さが環境負荷に直結する一方，エンジン停止状態で更新する場合は，電力供給可能な時間が限られる．

ディーラの整備員が専用ツールをつかって ECU 上のプログラムを書き換える従来のソフトウェア更新プロセスにおける処理時間の内訳を図 2-18 に示す．従来の更新プロセスで

は，車両外の診断装置から CAN を経由してプログラム全体を更新対象 ECU に送信し，ECU 上で Flash ROM を書き換えることでソフトウェアの更新を行う．図 2-8 に示す通り，ECU のソフト更新時間の大半は CAN による伝送時間が占める．ここでは，2MB のプログラムを 500Kbps の CAN の帯域のうち 10%を利用して伝送した場合の例を示した．

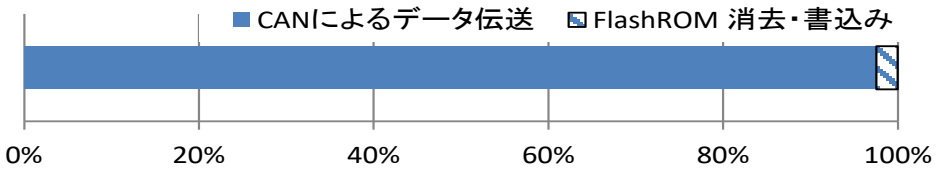


図 2-18 ECU のソフト更新時間の処理内訳

OTA によるソフト更新において，車載ゲートウェイへの配信までを走行中等に実施すると仮定すると，車両を利用できない時間は図 2-16 の更新適用処理の間である．更新適用処理は，診断装置が車載ゲートウェイに置き換わることを除いては従来の更新プロセスと同様であるため，処理時間の内訳は上記と同様になる．このため，OTA によるソフトウェア更新時間の短縮には CAN の伝送時間の短縮が課題となる．このようなプログラム更新のためのネットワーク伝送時間短縮技術として，差分更新技術がある[16][85][86][87][94].

清原[16]は，携帯電話のプログラムの遠隔更新に関して，Flash ROM の書換え領域の局所化による書換え時間の短縮及び，差分更新を用いた通信データ量削減による配信時間の短縮を提案している．前述の通り，ECU の書き換え処理においては，Flash ROM の書換え時間が占める割合は相対的に小さく，通信量を削減できる差分更新の時間短縮効果が高いと考えられる．プログラムの差分更新については他にも，商用ソフト[85][86]に加え，オープンソースソフトウェア(OSS)である bsdiff[87]が PC や携帯電話，カーナビに適用され広く利用されている．しかしながら，ECU に搭載されたメモリは PC や携帯電話と比較すると小さいため，これらのデバイス向けに開発されたソフトウェアをそのまま搭載することは困難であると考えられる．ECU と同様にメモリに制限のあるセンサネットワーク向けに差分更新を適用するため，差分抽出アルゴリズムと圧縮アルゴリズムの組み合わせを変えて差分サイズやメモリ使用量の評価がなされた研究[93][94]もあるが，アルゴリズムのデータ構造などを考慮した省メモリ化の手法は提案されていない.

Nakanishi[32]らは，前述の bsdiff で生成した差分を復元する際に，有向グラフを用いて復元時の書込み干渉を回避し，インプレース更新を適用してメモリ使用量を削減する方法を提案している．また，野澤ら[131][132]は，圧縮率向上を目的として，プログラムの変更によるバイナリモジュールの変化の内容に着目して bsdiff の差分抽出方法の改善を提案している．これらの方式はメモリ領域の削減や圧縮率の向上に有効な手段であると考えるが，

実際の車載 ECU のメモリサイズ等を考慮した搭載性評価がなされていない.

また，ECU 向けの更新時間の短縮に関しては，車内ネットワークとして FlexRay，CAN-FD や Ethernet を導入し，通信の広帯域化と複数 ECU の並列更新によって更新時間を短縮する方法も提案されている[133][134][135]．通信の広帯域化は更新時間短縮の有効な手段であるが，コストの観点からは，ソフト更新時間の短縮のみを目的として導入することは難しい.

本研究では，前述の研究を踏まえ，bsdiff をベースとして車載 ECU に搭載可能な省メモリな差分更新方式を提案する. また, 提案する方式を車載マイコン上に実装して評価を行い，メモリ使用量及び更新時間の短縮効果を確認する.

2.5. 結語

本章では，ネットワーク接続された組込みシステムの拡張に関する関連研究を体系的に整理した．最初に，システム拡張に必要な技術を，システム拡張を可能にする，または容易にする構成に関する技術とシステム拡張を実行する過程に関する技術に分類し，それぞれの先行技術を示した．システム拡張を可能または容易にする構成に関する技術においては，ハードウェアの再構成技術及び再構成を考慮したソフトウェアの構成に関する技術を示し，ソフトウェアの構成に関する技術としては，再構成の前提となる OS・バーチャルマシンレイヤの技術，再構成を容易にするミドルウェアやフレームワーク技術を整理した．システム拡張を実行する過程に関する技術においては，ノードの再構成のための更新データの生成と配送に関する技術と，追加または更新されたノードのシステムへの統合に関する技術について述べた．前者については配信データの作成技術，配信管理及びノードへの配信技術，ノードにおける更新適用技術について述べ，後者については，サービスの検出技術について述べた.

さらに，本研究の対象とするホームネットワークシステムと自動車システムについてその基本構成と関連技術を示し，システム拡張における課題をまとめた.

本研究では，以下，本章にて述べた課題を解決するための設計手法を提案する.

第3章 アプリケーション拡張性を考慮した標準プロトコル用ミドルウェア設計手法

3.1. 緒言

2.2.1 節では，ネットワーク接続された組込みシステムのシステム拡張を行う場合には，アプリケーションの拡張性を考慮した構成方法が重要であることを述べた．また，2.3.3 節では国内の HEMS 標準プロトコルとして推奨されている ECHONET Lite のミドルウェア化にあたっては，下位通信層が定義されていないといった前身プロトコルである ECHONET との差異や，組込みシステムの共通課題であるリソース制約について考慮した設計が課題であることを示した．

そこで本章では，これらの課題に対して，サービスゲートウェイ（SGW）向けの ECHONET Lite に準拠したコントローラ用ミドルウェアを OSGi バンドルとして構成する方法の提案を通じて，アプリケーションの拡張を容易にする標準プロトコル用ミドルウェアを構成するための設計手法について論じる．

3.2. サービスゲートウェイの ECHONET Lite バンドルの要件

本研究では，以下の 4 つの要件を満足するソフトウェアを開発することを目標とする．

R1 :アプリケーションの開発工数を低減できること．
R2 :省リソースとアプリケーションの停止時間最小化を両立できること．
R3 :機器の識別に必要なアプリケーションの実装量を低減できること．
R4 :様々な下位通信層に柔軟に対応できること．

以下にこれらの要件に関する考察を述べる．

3.2.1. サービスゲートウェイ上のミドルウェア実装における要件

R1, R2 は, サービスゲートウェイ上にミドルウェアを実装するときの要件である. 以下, 各要件の詳細について述べる．

R1：アプリケーションの開発工数を低減できること

本提案のミドルウェアバンドルを用いるアプリケーションの主要機能は，ECONET Lite

機器を操作（制御及び情報取得）して，デマンドレスポンスや見える化のようなサービスを提供することである．アプリケーション開発者にとってはサービス提供以外の機能の開発工数をできるだけ低減できることが重要である．そのため，本提案のミドルウェアバンドルにおいては，複数のアプリケーションに共通的に利用される機器発見などの機能を適切に構成し，アプリケーションの開発工数をできるだけ低減できるようすることが課題となる．本研究では，従来 ECHONET で定義された Java API[28]を利用する場合と比較してアプリケーションの開発工数を低減できることを目標とする．

R2：省リソースとアプリケーション停止時間最小化を両立できること

サービスゲートウェイ上のソフトウェア実装にあたっては，省リソースであることと，搭載されたアプリケーションが提供するサービスを停止させないようすることが重要である．これは，サービスゲートウェイは，組込みシステムとして構成され，省リソースが要求される場合が多いこと，さらに，マルチサービスを実現するサービスゲートウェイにおいては，アプリケーションの要求によりサービスゲートウェイそのものや ECHONET Lite ミドルウェアの無停止を求められるケースもあるためである．

以上から，ECHONET Lite バンドルでは，省リソースかつアプリケーションの停止時間を最小化できる構成を実現することが課題となる．本研究では，ECHONET Lite の Java 実装[114]と比較して省リソースな構成で停止時間を 0 秒にできることを目標とする．

3.2.2. ECHONET Lite 規格の実装上の要件

R3,R4 は，ECHONET Lite 規格を実装するにあたって，規格の課題に対応するための要件である．以下に，各要件の詳細について述べる．

R3：機器の識別に必要なアプリケーションの実装量を低減できること

ECHONET Lite を用いて接続された機器をユーザが利用するにあたって，プロトコルや下位通信層で定められた識別情報のままではどの機器がどれに当たるか分かりにくい．そのため，例えば「リビングのエアコン」といった機器名称などの人間に分かりやすい識別情報と紐づけ，ネットワークの構成や機器のアドレス情報などが変更されてもそのまま対応付けができる必要がある．例えば，UPnP では UUID（Universally Unique Identifier）と機器名称を紐づけておけば，IP アドレスが変わっても接続された機器が一意に識別できる．一方， ECHONET Lite 規格では機器を永続的に一意に識別するためのいくつかの方法は

提供されているが，どの情報を利用可能にするかはメーカの実装依存となっており，すべての ECHONET Lite 機器で統一的に利用できない．表 3-1 に ECHONET Lite 規格において機器識別に利用できる情報と当該情報を利用するにあたっての課題を示す．[136]では本課題について検討し，個体識別番号あるいはメーカコードと製造番号の組み合わせのどちらかを使うと一意に識別（トレース）可能という考察がなされている．しかし実際はいずれの方法も課題があり，すべての機器で統一的に利用できる方法がないことから，コントローラ用のミドルウェアは，アプリケーションが ECHONET Lite 機器を容易に一意に識別する仕組みを提供することが課題となる．

本研究では，Java API[28]に基づく実装と比較して，アプリケーションが操作対象の機器選択のために必要な実装量を低減できるような機能を提供することを目標とする．

表 3-1 ECHONET Lite における機器識別のための情報

識別情報	内容	課題
下位通信層における識別情報	IP アドレスまたは MAC アドレス	下位通信層のプロトコル・接続方法に依存
識別番号	オブジェクトをドメイン内で一意に識別するための情報（8～16byte）	v1.01 では，0x00(未設定)も設定可能
個体識別情報	ドメイン内で各ノードを一意に識別可能とし，かつ機器の移動（サブネットの変更など）後も常に同一ノードは不変なものとして取扱い可能とするための情報（2byte）．初期値は任意（乱数など）．コントローラから変更可能	コントローラが複数存在する場合の競合回避方法などのガイドライン無し
メーカコード+製造番号	機器の製造者が一意であることを保証する番号	製造番号はオプション

R4：様々な下位通信層に柔軟に対応できること

ECHONET Lite ではプロトコルの軽量化とグローバルな標準仕様を利用したいという要望に基づいて OSI 参照通信レイヤ 4 層以下の下位通信層は規格範囲外とされた．しかし一方で下位通信層に規定がないことは，伝送メディアや下位通信層の実装の相違による相互接続性の問題につながる可能性がある． ECHONET Lite の下位通信層の候補としては UDP/IP/Ethernet 以外にも複数の通信層（920MHz 帯無線を利用する ZigBee™ IP など）が想定されている[137]．ECHONET Lite を使用するシステムでどの下位通信層を採用するかはユーザまたは機器ベンダの選択によるため，家毎，機器毎に多様な構成となってしまう．また，図 3-1 に示すようにサービスゲートウェイは複数の下位通信層にまたがる機器と接続するというユースケースも考えられる．そのため，さまざまな機器と接続する可能性

のあるサービスゲートウェイは，複数の下位通信層に柔軟に対応できる構成である必要がある．ここでいう柔軟とは，下位通信層の入れ替えまたは追加のために，ECHONET Lite バンドルの改変が不要であること，及び，ECHONET Lite バンドルを停止することなく下位通信層の追加または入れ替えができること，を意味する．下位通信処理部の実装方法によっては，家毎の環境に合わせて複数のバージョンのバンドルが必要となり，開発効率や保守性に問題が生じる．また，新しい機器を導入し，新しい下位通信層に追加対応する場合にはバンドルの更新が必要になり，その間アプリケーションを停止させてしまう可能性がある．そこで，本研究では，バンドルを改変せず，かつバンドル及びアプリケーションを停止せずに下位通信層の追加や変更が可能な構成を実現することを目標とする．

図 3-1 2 つのサブネットにまたがる HEMS の例

3.3. ECHONET Lite バンドルの設計

ECHONET Lite バンドル及び関連するバンドルについて，図 3-2 の構成を提案する．すなわち，本提案の構成では，アプリケーションは機器操作 I/F 抽象化バンドル，ECHONET Lite I/F 実装ドライババンドルを介して，ECHONET Lite バンドルを利用する．機器操作 I/F 抽象化バンドル及び ECHONET Lite I/F 実装ドライババンドルは，機器オブジェクト毎に準備し，機器オブジェクトのプロパティ定義はそれぞれの ECHONET Lite I/F 実装ドライババンドルが保持する．プロパティ定義とは，ECHONET Lite において，具体的に何をどう制御するかをやり取りするための定義情報である．ECHONET Lite バンドルは，自機器管理機能，他機器管理機能，通信処理機能から構成する．自機器管理機能では，コントローラの機器オブジェクト及びノードプロファイルオブジェクトのインスタンスを管理する．また，他機器管理機能は，発見した機器オブジェクトのインスタンスを管理する．これらのインスタンスを利用する場合は，機器操作インタフェースを利用する．ネットワークへの電文送出と受信を行う下位通信層バンドルは，下位通信層の種類毎に準備し，下位通信層に依存する処理を実装する．電文の送受信は，ECHONET Lite バンドルの通信処理部の受信用インスタンスと下位通信層バンドルの送信用インスタンスが，下位通信層抽象化インタフェースを介して行う．

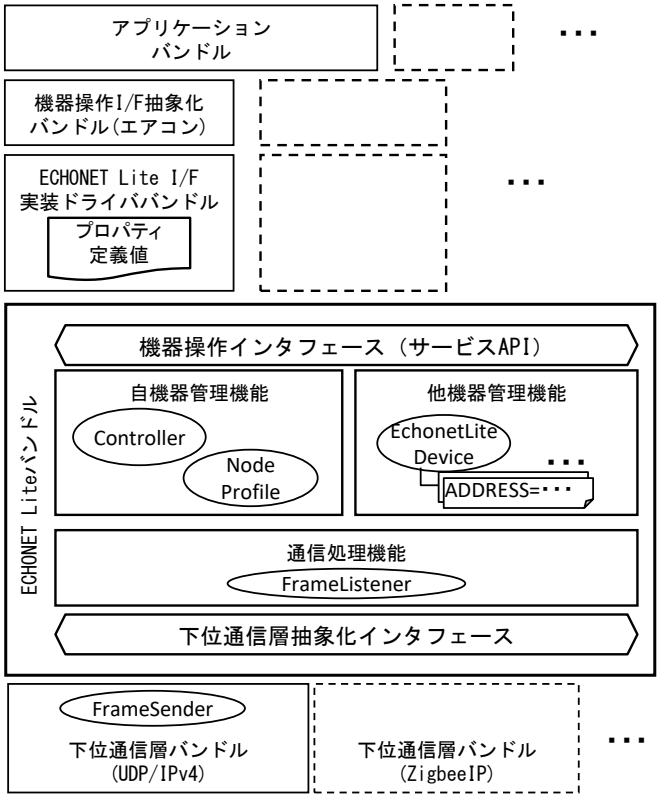


図 3-2 ECHONET Lite バンドルとその周辺構成

以下に，前述の要件と，提案するバンドル構成についての詳細を述べる．

3.3.1. アプリケーション開発工数低減への対応

アプリケーション開発工数低減の観点から，ECHONET Lite バンドルが提供すべき機能として，他機器管理機能，自機器管理機能，通信処理機能を提案する．

本提案のバンドルを搭載するサービスゲートウェイの基本的なユースケースとしては以下のようなステップが考えられる．

(a)ネットワーク上に接続された機器を探索・発見する
(b)発見した機器から操作対象を選択する
(c)選択機器に対して，操作要求を発行する．
(d)要求に応じて ECHONET Lite 電文を構成し，ネットワークに送出する．
(e)機器からの応答や通知を受信し，電文を解析する．
(f)前記解析結果に基づいて要求の成否を判定し，結果に応じた処理を行う．
(g)他機器からの ECHONET Lite 電文による要求を受信し，応答する

アプリケーションは，上記(a)から(f)のステップが実行できれば，「使用電力をスマートメータから取得し，表示する」ことや「取得した室温や使用電力に基づいてエアコンの設定温度を変更する」などのサービス実現が可能になる．

上記(a)から(f)のステップのうち，(b)(c)(f)のステップ，すなわち，ネットワークに接続された機器のうち，どれを選択してどのような要求を行い，その結果に応じてどのような処理を行うかについては，アプリケーションごとに異なる．一方で，(a)(d)(e)は複数のアプリケーションが共通的に必要とする機能である．このうち，(d)(e)はプロトコルを実装したミドルウェアが通常提供すべき機能であるため，本提案においても通信処理機能として提供する．また，(a)に関し，ECHONET Lite 規格は，プラグアンドプレイにより機器の発見ができるという特徴を有している．しかし，機器発見の具体的な手順については規定されていないためアプリケーション開発者が個別に設計及び実装を行う必要がある．そのため，本提案のバンドルにおいて，手順(a)の処理を他機器管理機能として提供する．

また，本提案のバンドルを搭載するサービスゲートウェイはコントローラであるため，他の機器から情報を取得されることや制御されること（前記(g)に相当）は想定する必要がない場合が多い．一方，ECHONET コンソーシアムによる規格適合性認証[138]では，「送信電文は適切か」「受信電文を正しく解釈し，適切に自機器・自ノードオブジェクトの処理ができているか」という被制御機器としての項目が中心となっている．これを実現するためには，自機器・自ノードオブジェクトの管理が必要になる．このような自機器・自ノードオブジェクトの管理はミドルウェアの範囲外として，アプリケーション側での対応が必要とされる場合が多い[5]．そこで，本提案のバンドルでは，(g)に対応し，被制御機器としての役割を担う自機器管理機能を提供する．図 3-3 に電文受信時のシーケンスの比較を示す．

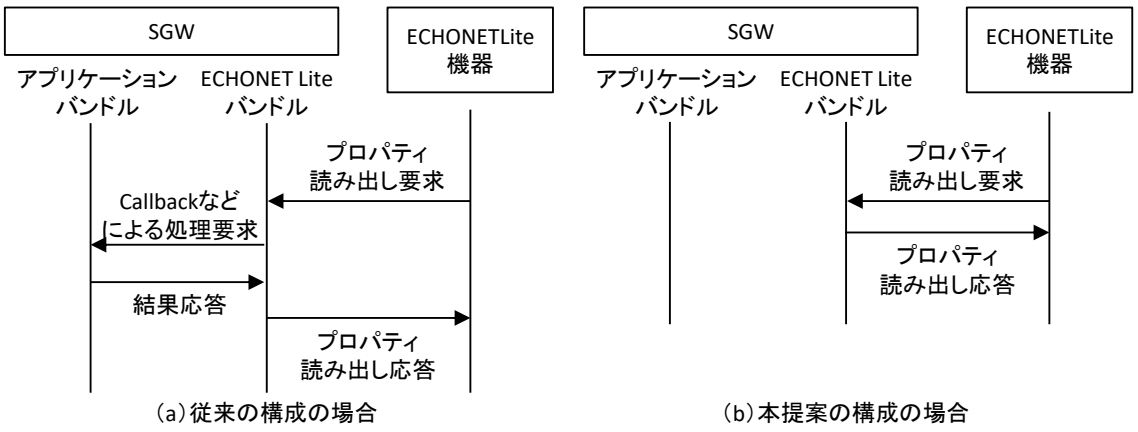


図 3-3 要求受信時のシーケンス比較

以下に，各機能の詳細を述べる．

通信処理機能

(d)(e)に相当する電文生成や解釈といったプロトコル処理に加え，不正な電文を発行しないためのチェックや，不正な電文を受信した場合の異常処理などを行う．

他機器管理機能

前述の(a)ECHONET Lite 機器の探索・発見を行う．また，発見した機器をアプリケーションから利用可能な OSGi サービス（ECHONETLiteDevice サービス）として OSGi フレームワークに登録する（図 3-4）．さらに，応答状態を監視し，応答がなくなった機器は OSGi サービスから削除する．アプリケーションは，ECHONETLiteDevice サービスが提供する機器操作インタフェース（サービス API）を利用することで，発見した機器を操作したり，イベントを受信したりすることができる．ECHONET Lite バンドルは，機器操作インタフェースによるアプリケーションからの要求（API コール）に従ってネットワークに電文を送出し，対応する応答を API の戻り値やイベントとしてアプリケーションに通知する．

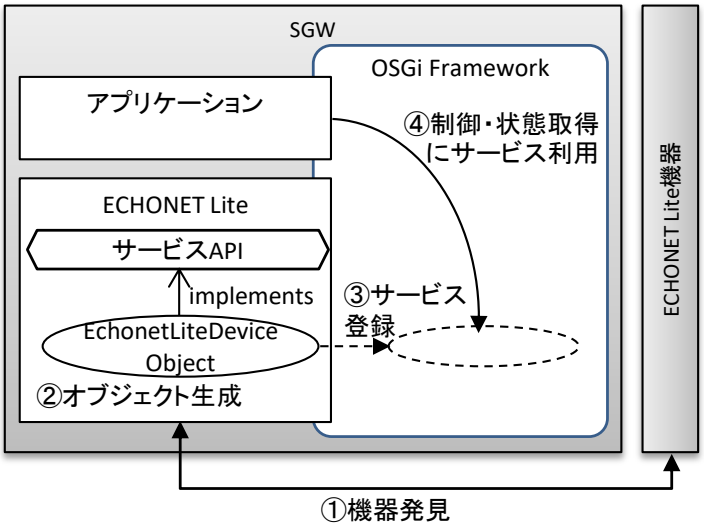


図 3-4 OSGi を利用した機器登録・利用方法

以上のように，本提案ではサービスゲートウェイのプラットフォームとして，プラグアンドプレイによる機器発見及び機器管理機能や，発見した機器に対する操作インタフェースをアプリケーションに提供する．これにより，アプリケーションは所望の機器を制御するためには OSGi サービスの検索・取得のみ実装すればよく，独自の制御アルゴリズムなど，ユーザにサービスを提供するための機能 の実装に注力することができると考える．

自機器管理機能

コントローラとして搭載が必要な機器オブジェクトスーパークラスの及びノードプロファイルクラスの必須プロパティを管理し，読み出し要求に対する応答，書き込み要求に対する管理値の更新や状態変更通知を行い，被制御機器としての(g)に相当する機能を提供する．これは，前述の通りアプリケーションは接続された機器の操作が主眼であるため，自機器・自ノードオブジェクトの管理を最小限にできることが望ましいためである．本機能により，アプリケーションは自機器について特別な管理をしなくても，ECHONET Lite 規格でいうコントローラとしての最小限の要求を満足し，サブネット内の他機器からは，一つの ECHONET Lite「コントローラ」機器として認識される．本機能により，アプリケーション開発者は ECHONET Lite コントローラ機器としての認証取得のための工数を低減できる，と考える．

3.3.2. 省リソースとアプリケーション停止時間の最小化への対応

本提案の構成では, 省リソース化とアプリケーション停止時間の短縮のため, ECHONET Lite で定義された機器の ECHONET プロパティ定義の値は保持せず, 外部でバンドル化して保持することとする．

ECHONET Lite では多様な機器操作のための情報が定義されており，これは他の規格と比較した場合の ECHONET 規格の強みである．一方で，これらを最初からフル実装した場合, 定義値の数だけプログラムサイズが大きくなってしまい, 大きなリソースが必要となる．これを解決し，省リソースを実現する１つの方法として，必要最小限の機器オブジェクト定義のみ実装し，必要に応じてソフトウェアの更新を行って順次追加する方法がある．しかしこの場合，サービスゲートウェイまたはソフトウェアの再起動が必要となり，無停止が要件であるアプリケーションには都合が悪い．

また，前述の機器操作インタフェースはこの定義値をどこで管理するかによって 2 通りの定義方法が考えられる．すなわち，ミドルウェアのレイヤで定義値を管理して操作毎の API（setTemperature()など）を定義しアプリケーションは ECHONET 規格の定義値を意識しないようにする方法（図 3-5（a））と，アプリケーションが ECHONET 規格を意識し定義値を管理してプロパティの値を渡す全操作に共通の API（setProperty()など）を定義する方法（図 3-5（b））である．

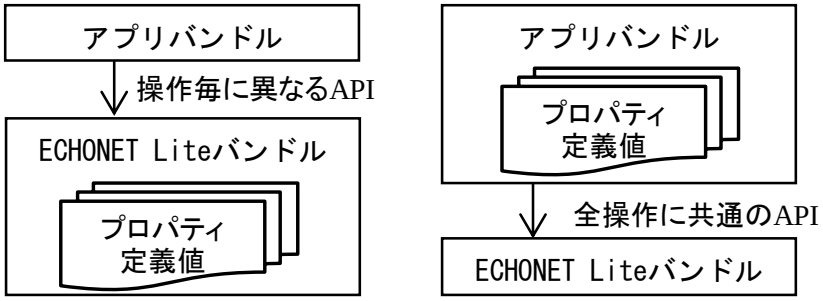


図 3-5 機器操作インタフェースの提供方法

表 3-2 に，省リソースとアプリケーション停止時間の観点から各方法を比較した結果を示す.

比較の結果，省リソースとアプリケーション停止時間の最小化を両立するためには，プロパティの定義値をバンドル外で管理し，必要に応じて順次追加していく方法がよいと考えられる．ただし，このままでは，更新対象のアプリケーションは更新時に停止する必要があることと，アプリケーション開発者は ECHONET Lite プロトコルの詳細定義を意識する必要が生じてしまうという課題がある.

そこで，本バンドルを利用するアプリケーション周辺の構成として，図 3-6 の(b)’の構成を提案する.

表 3-2 機器操作インタフェース定義方法の比較

実装方法		省リソース	停止時間
(a)	フル実装	× 全ての機器のプロパティ定義値を保持	○ あらかじめ全て実装されているため更新不要
	順次追加	○ 必要最小限のプロパティ定義値のみ保持	× 追加時にミドルウェア停止・更新が必要
(b)	フル実装	× 全ての機器のプロパティ定義値を保持	○ あらかじめ全て実装されているためアプリ更新不要
	順次追加	○ 必要最小限のプロパティ定義値のみ保持	△*1 制御対象を追加する際に当該アプリケーションの停止・更新が必要

*1 更新対象以外のアプリケーションには影響しない.

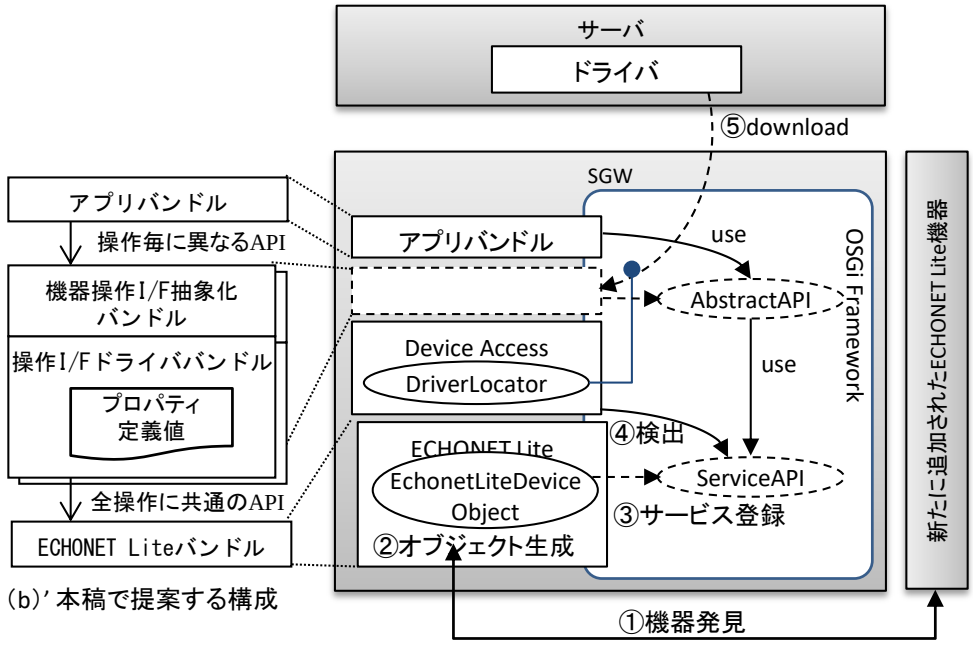


図 3-6 機器操作インタフェースの実装方法とそれを利用した抽象化インタフェースの構成

具体的には，アプリケーション向けインタフェースとして別途操作内容を抽象化したインタフェース（機器操作 I/F 抽象化バンドル）を定義し，これを実装した機器操作 I/F ドライババンドルを用意する．プロパティ定義値をこのドライババンドルで保持し，ECHONETLiteDevice サービスを呼び出すことにより，開発者は ECHONET Lite 規格の詳細を知らなくてもアプリケーションを開発できるようになる．さらに ECHONETLiteDevice サービスを Device Access[139]に準拠させておく．これによって，新しい種別の機器を ECHONET Lite バンドルが発見したときに自動的に対応するドライバを検索, 及びダウンロードできる仕組みが容易に構成できる. Device Access サービスは，機器を表現する Device サービスと，機器に接続するための実装を担う Driver サービス等から構成されこれらを結合する機能を提供する．また，適切な Driver サービスがフレームワーク内に存在しない場合，サーバからこれを取得する機能も提供する．ここでは，ECHONETLiteDevice サービスを Device サービス，機器操作 I/F ドライバを Driver サービスとすることで，新規機器に対応するドライバの自動取得を実現できる.

以上の構成とすることにより，各家庭の機器状況に応じた最小限の構成とし，省リソースを実現しつつ，機器追加時のアプリケーション停止時間を最小限にすることができると考える.

3.3.3. 機器の識別に必要なアプリケーションの実装量低減への対応

本提案の構成では，機器識別のために必要なアプリケーションの実装量低減を実現するため，アプリケーションが機器の識別情報として OSGi においてサービス検索に用いるフィルタ文字列を利用し，提案の ECHONET Lite バンドルは検索に必要な情報を予め機器から取得しておく構成とする．

前述のとおり，ECHONET Lite 規格では，機器を一意に識別する統一的な方法が規定されていない．このため，アプリケーションは，機器を一意に識別するために機器毎に異なる情報を用いる必要が生じる可能性がある．例えば，機器 A は識別番号での識別を想定して，識別番号に有意な値が設定されるが，機器 B は MAC アドレスでの識別を想定して識別番号は 0 であるといったケースが考えられる．この場合，ユーザ（または設置者）は機器外装に印刷された識別番号や MAC アドレスを使って機器登録を行い，ユーザフレンドリーな情報（機器名称など）との紐付けを行うというユースケースが考えられる．このようなユースケースを実現するためにはアプリケーションは，ユーザが登録した情報に応じて機器から必要な情報を取得し，かつそれをなんらかのデータベース（DB）にて管理する必要が生じる．この際，機器を識別するための情報が一意ではないため，情報登録や検索の方法が複雑になってしまう可能性がある．

一方，OSGi フレームワークでは，登録するサービスにサービスプロパティと呼ばれる付加情報を設定しておくことで，登録されたサービスの検索及び追跡にフィルタ機能を利用することができる．このフィルタは LDAP アプリケーションフィルタ[140]と呼ばれるフィルタであり，複数の条件を含む複雑な検索式を構成することができる．そこで，本バンドルを利用するアプリケーションが，ECHONETLiteDevice サービスを取得する際に，このフィルタ機能を活用することができるように機器を一意に識別するために利用される可能性のある情報を予め取得し，サービスプロパティとして登録しておくこととする（図 3-7）．

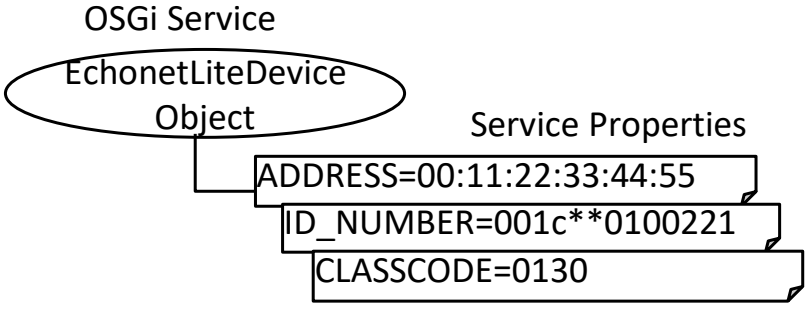


図 3-7 ECHONETLiteDevice サービスのプロパティ構成

このようにECHONET Lite機器の情報をOSGiのサービスとして管理することで, OSGiフレームワークが提供するサービスレジストリの仕組みを機器管理DBとして利用できる. さらに, 機器管理DBのインデックスとしてECHONETプロパティ情報を利用できるようにすることで, アプリケーションが機器を容易に識別できるようになると考えられる. すなわち, アプリケーションは, 機器ごとに異なる可能性のある情報の取得や管理・比較などの処理を実装する必要はなくなり, 機器を表すECHONET Liteサービスを取得するためのフィルタ文字列を識別情報として管理するだけでよい.

3.3.4. 様々な下位通信層への柔軟な対応

ECHONET Liteバンドルでは下位通信層との結合を疎なものにするため, 下位通信層とのインタフェースとして下位通信層の構成に依存しない抽象的なネットワークインタフェース（下位通信層抽象化インタフェース）のみ規定する. すなわち, 下位通信層を実現するソフトウェアは ECHONET Lite バンドルの外部で前記インタフェースを実装したバンドル（下位通信層バンドル）とする構成をとる（図 3-8）. 通信用リソース（ソケットやCOMポート）の確保, 各種プロトコルやデバイス実装に依存したパケット化などは, 下位通信層バンドルにて対応する. これにより, 下位通信層とECONET Liteミドルウェア バンドルを切り離し, 異なる物理層のネットワークインタフェースが複数存在する場合も対応可能にする. 本構成によって, 下位通信層を変更・追加する場合でもECHONET Liteバンドルの修正を行うことなく, 新しく定義された下位通信層に対応することができる.

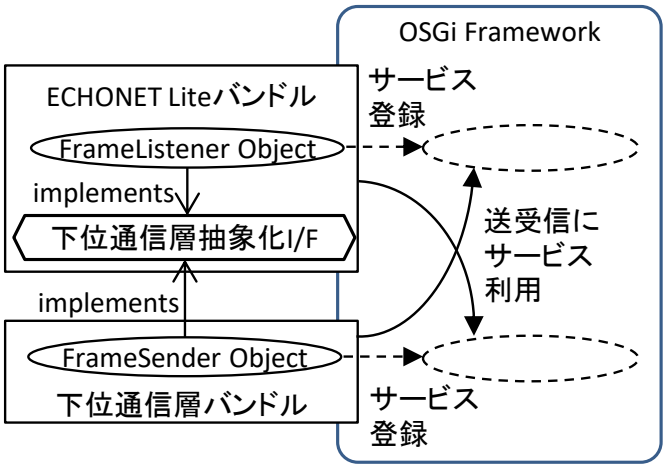


図 3-8 下位通信層バンドル構成

さらに, OSGiのバンドル追跡機能を利用して, 下位通信バンドルの新規登録を追跡する

ことで， ECHONET Lite バンドルを停止することなく，新しく追加された下位通信バンドルを利用して新しい下位通信層に接続された機器を発見・制御することが可能となると考えられる.

また，新規追加されたネットワークに対応した下位通信層バンドルをサーバからダウンロードして追加することも可能となる. この場合のシーケンス例を図 3-9 に示す. 例えば，サービスゲートウェイの USB ポートに ZigBee IP スタックを搭載したドングルを挿入した際，対応した下位通信バンドルをダウンロードして実行する仕組みを作り込んでおくことで，以降，当該インタフェース（USB-ZigBee IP）を経由した機器の操作が可能になる.

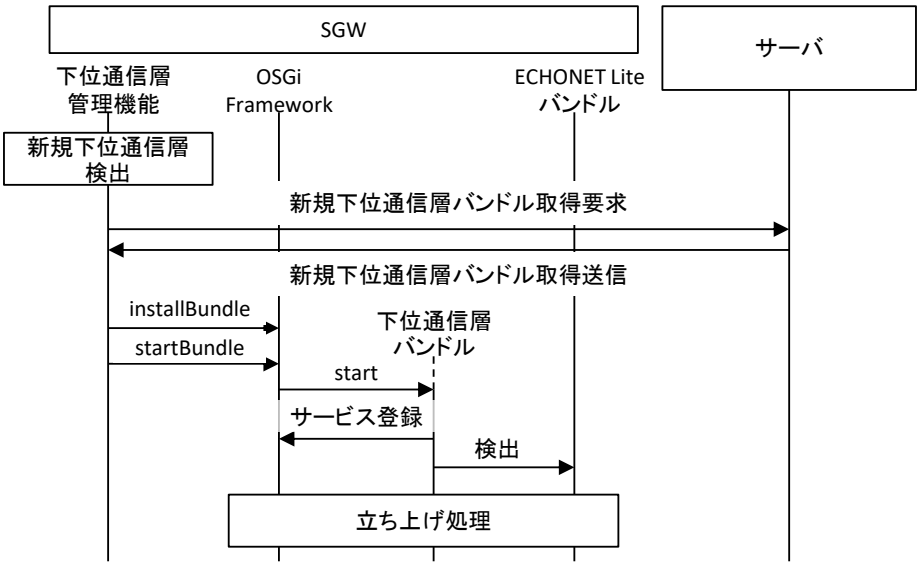


図 3-9 新規インタフェース追加時のシーケンス

以上の構成により，本提案のバンドルを更新せず異なる下位通信層を選択・追加できるとともに，サービスゲートウェイのように連続運転が要求される機器においても，機能を停止することなく新しい下位通信層を導入することが可能になる.

3.4. 実装と評価

3.3 節で提案した ECHONET Lite バンドルを PC 及びサービスゲートウェイ実機上で試作し，評価を行った．表 3-3 に評価に用いた機器の主要諸元を示す．また，図 3-10 に評価用システムの概要を示す.

表 3-3 試作機器仕様

項目	サービスゲートウェイ	PC
CPU	650MHz	2.7GHz
Memory	RAM:512MB, ROM:128MB	RAM:8GB HDD:280GB
LAN interface	4port(LAN), 1port(WAN)	1port
USB interface	2port	4port
OS	Linux	Windows7 64bit
Application Platform	JavaME (SuperJ Engine™)/ OSGi Framework (SuperJ Engine Framework™)	J2SE6 OSGi Framework (SuperJ Engine Framework™)

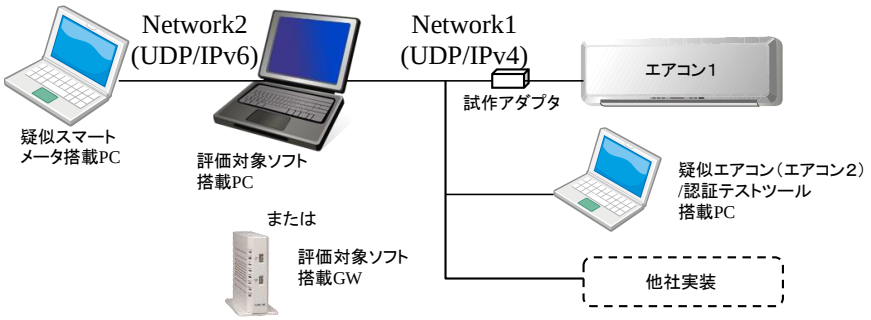


図 3-10 評価システム概要

実装した試作ソフトウェアを ECHONET コンソーシアムの主催するプラグフェストにて他社実装と接続した．ECHONET Lite バンドル単体で機器発見し，OSGi フレームワークに接続された機器のサービスが登録されること，及び取得した ECHONET プロパティが前記サービスのサービスプロパティとして登録されていることを確認した．これにより，実装したバンドルのプロトコル実装が相互接続性を確保できていることが確認できた．なお，PC とサービスゲートウェイ上では，下位通信層バンドルの一部を除いて実装の変更なくバンドルを動作させることができた．

3.4.1. サービスゲートウェイ上のミドルウェア実装における要件の評価

R1：アプリケーションの開発工数を低減できること

表 3-4 に，アプリケーションの開発工数低減を目的として ECHONET Lite バンドルに実装した他機器管理機能，自機器管理機能の概算ステップ数とステップ数から概算した工数を示す．ここに示すとおり，前述の構成とすることで，アプリケーション毎にこれらの機能

の実装をする必要がなくなり，[28]のようなアプリケーションにこれらの機能を実装する場合と比較して 1 アプリケーションあたり約 2.4 人月の工数を低減できる．

表 3-4 共通機能の工数概算

機能	Step 数	工数(人月)
他機器管理機能	480	1.1
自機器管理機能	570	1.3

また，自機器管理機能を利用することで，実装工数に加えて適合性認証に係る工数も低減できることを確認するため，認証項目についての評価を行った．表 3-5 に ECHONET Lite における規格適合性認証の自己認証の項目数を示す．コントローラとして適合性認証を取得するためには，少なくとも 88 の必須項目の確認を行う必要がある．このうち，ほぼ 2/3 の項目が ECHONET Lite フレーム受信時の要求仕様に関するものである．前述のように本バンドルでは自機器管理機能を備えているため，アプリケーションは改めて何らかの実装を行うことなく，大半の項目に適合させることができる．電文送信に関しても，不正な電文を送信しない機能を提供することから，アプリケーションに依存して（アプリケーション追加・変更時に,）再テストが必要な項目は他機器への SET 要求送信に関する 4 つとなる．これらは，他機器オブジェクトに送信するプロパティの整合性確認及び不正チェックを行う項目である．

表 3-5 ECHONET Lite における自己認証試験項目数

	全体	必須	受信	送信	
				自	他
フレーム処理	31	30	15	15	
オブジェクト処理	98	58	52	2	4
合計	149	88	68	20	

他機器オブジェクトのプロパティ値の管理は，バンドル外で行うこととしたため，これらの認証項目については，本提案のバンドルでは担保できない．しかし，自機器管理機能を利用することによって 88 項目中 4 項目を除いて，バンドルの提供する機能で対応できるため，本バンドルを利用した場合，利用しない場合と比較して適合性認証に係る 84 項目分の開発工数を低減できる．

以上から，本提案の構成とすることで接続機器の管理などをアプリケーションが実装する構成[5]と比較して，アプリケーション開発工数を低減するという要件 R1 は満足できたと言える．

R2 ：省リソースとアプリケーションの停止時間最小化を両立できること

省リソースな実装であることを評価するため,実行モジュールのサイズを比較した.また,アプリケーション停止時間を評価した．表 3-6 に結果を示す.

すべての機器オブジェクト及びプロパティを予め実装している[114]の構成(A)では，モジュールサイズが約 1,200KB と大きなものとなっている．一方，本提案(C)では，家庭用エアコンクラスのみを含む場合の構成で，約 162KB でモジュールを構成でき，省リソースであると言える．[114]において，対応するオブジェクトを家庭用エアコンのみにしてモジュールを構成した場合(B)，約 95KB と省リソースの構成にできるが，Java 実装においてソフトウェアを更新する際にはミドルウェアの停止が必要になるという課題が残る.

表 3-6 モジュールサイズとアプリケーション停止時間

機器オブジェクト定義実装方法	ROM サイズ (KB)	AP 停止時間(秒)
(A)すべてを予め実装*1 （従来の Java 実装）	× (約 1200:[9])	○ (0)
(B)必要に応じ追加*2 （従来の Java 実装）	○ (約 95:[9])	× (1.7～86)
(C)必要に応じ追加*2 （本提案の実装）	○ (約 162)	○ (0)

*1 全機器オブジェクトを含む場合
*2 家庭用エアコンオブジェクトのみの場合

そこで，次に，従来の Java 実装(B)において機能（機器オブジェクト定義）追加の時のアプリケーション停止時間について見積もるため，システムの再起動に要する時間及び，バンドルの更新（ここでは，ECHONET Lite バンドルが停止直前の状変通知を発行してから，再起動してインスタンス通知を発行するまでの）時間を計測した．その結果，システムを再起動に要する時間は約 86 秒，バンドルを更新に必要な時間は，1.7 秒であった．これらの結果から，通常の Java 実装でソフトウェアを更新時のアプリケーション停止時間は，バンドル更新時間 1.7 秒からシステム再起動 86 秒の範囲の時間と評価した．一方で，バンドルを追加するだけの(C)の場合，アプリケーション停止時間は 0 秒で，新しい機能を追加することが可能であり，従来の Java 実装と比較してアプリケーション停止時間の短縮を実現できる.

以上の通り，提案の構成(C)とすることで，従来の Java 実装(A)(B)と比較して，省リソースとアプリケーション停止時間の最小化を両立するという要件 R2 は満足できたと言える.

3.4.2. ECHONET Lite 規格の実装上の要件に対する評価

評価システムを利用し，以下のシナリオで要件 R3 と R4 についての評価を実施した．なお，評価用 GW は試作機で IPv6 未対応のため，評価は PC 上で実施した．

<前提条件>
・ユーザ宅の既存ネットワーク 1 にはエアコン 1，エアコン 2 が接続
・エアコン 1，2 は DHCP でアドレスを取得
・サービスゲートウェイとエアコン 1，2 は UDP/IPv4 で通信

<シナリオ>
①ユーザは 2 つのエアコンを区別しやすいように名称を登録
エアコン 1：リビングエアコン
エアコン 2：寝室エアコン
②登録にあたって，ユーザが知ることができる情報
エアコン 1：MAC アドレス
エアコン 2：識別番号
③サービスゲートウェイ上のアプリケーションは，エアコンから室温を取得し，前記名称とともにリアルタイム(1 秒更新)でユーザに提示.
④ユーザ宅に新規にスマートメータ(SM)が導入され，新規ネットワーク（UDP/IPv6）に接続
⑤サービスゲートウェイは SM からの電力量をリアルタイムでユーザに提示.

R3:機器の識別に必要なアプリケーションの実装量を低減できること

上記シナリオにおいて，機器を特定する情報として異なる識別情報を登録した 2 つの機器を正しく識別し，情報を取得できた．表 3-7 に本シナリオにおけるアプリケーションの機器管理テーブル例を示す．このようなフィルタ文字列を用いて機器操作サービスを取得することで，利用する識別情報が異なる場合でも，OSGi フレームワークからサービスを取得する手順に従って，10 ステップ程度の実装のみを行えば目的とする機器の識別・選択ができることを確認した.

本提案の機能を用いない[28]のようなインタフェースの場合，アプリケーションは個別に機器毎に異なる可能性のある識別情報の取得処理や管理を行なわなければならない．これ

らの実装量は，本提案の他機器管理機能と同程度と仮定すると，480 ステップ程度となる．また， OSGi のフィルタ機能は約 1200 ステップで実装されている．一方で，本提案の手法では前述の機能を利用することによって，アプリケーションは OSGi を利用する際は必ず実装するサービス取得処理と簡単な識別情報の管理のみ実装すればよい．すなわち，前述の他機器管理機能及びフィルタ機能の全てあるいはその一部を実装する必要がある従来の手法と比較して，少ない 実装量で機器の識別ができる．以上から，[28]に基づく実装と比較して，機器選択のために必要な実装量を低減できる機能を提供するという要件 R3 は満足できたと言える．

表 3-7 OSGi のフィルタを利用した機器管理テーブル

機器名称	検索フィルタ
リビングエアコン	(ADDRESS=00:11:22:33:44:55)
寝室エアコン	(ID_NUMBER=FE00000100****03)

R4:様々な下位通信層に柔軟に対応できること

シナリオの④では，手動で IPv6 に対応した下位通信層バンドルを新規追加し，すでに接続されたエアコンからの情報取得が途切れないこと，を確認した．すなわち，本提案のバンドルを改変する追加工数 0 かつ，アプリケーションの停止時間 0 秒で，新しい下位通信層に対応できた．これにより，複数の異なる下位通信層が存在する環境でも ECHONET Lite バンドルを改変することなく，かつバンドル，アプリケーションを停止することなく下位通信層を追加する構成を実現するという要件 R4 は満足できたと言える

3.5. 結語

本章では，国内の HEMS 標準プロトコルである ECHONET Lite に準拠したコントローラ用の OSGi バンドルについて検討し，PC 及びサービスゲートウェイ上に実装して評価を行った．設計にあたっては，マルチアプリケーションを実現するサービスゲートウェイ上のミドルウェアとして実装するための課題及び規格実装上の課題に着目して 4 つの要件を抽出し，これを基に ECHONET Lite バンドル及びその周辺バンドルの構成を提案した．その際，OSGi フレームワークの持つ機能を最大限利用できるよう考慮した．その後，サービスゲートウェイ上に設計したバンドルを実装，評価し，提案する構成の有効性を確認できた．

また，本研究において開発した ECHONET Lite バンドルは，2013 年より数百戸の集合住宅に導入され，利用されている．

以上により，OSGi に代表される，動的なソフトウェアモジュールの再構成機構を持つプ

ラットフォーム上で，アプリケーションの開発工数を低減して拡張を容易にするための標準プロトコル用ミドルウェアの設計手法を示した．具体的には，プロトコル依存の機器個別プロパティ情報をミドルウェアとは別のソフトウェアモジュールとして機器種別ごとに構成すること，プロトコル規定外の下位の通信層も同様に別のソフトウェアモジュールとして構成すること，上記ソフトウェアモジュールとミドルウェア間のインタフェースを抽象化してプロトコル依存をなくすこと，により，アプリケーション開発工数を低減しつつ，組込みシステム共通の課題であるリソース制約への対応と可用性確保が可能なシステムの構築が可能であることを示した．

第4章 システム多様性を考慮した遠隔ソフトウェア更新制御機能の設計手法

4.1. 緒言

2.2.2 節では，ネットワーク接続された組込みシステムのノードを再構成してシステム拡張を行う場合に，多様性のあるノードから構成されるシステムの更新制御が課題であることを述べた．また，2.4.3 節では自動車システムの更新制御にあたっては，多様な ECU からなるシステムの更新を制御するための機能要件の整理とその機能配置，多様性に対応した上で，組込みシステムの共通課題であるリソース制約について考慮した設計が課題であることを示した．

そこで本章では，これらの課題に対して，軽量スクリプトを用いて自動車システムの遠隔ソフトウェア更新制御として構成する方法の提案を通じて，更新手順などが異なる多様なノードから構成されるシステムを拡張するための更新制御機能の設計手法について論じる．

4.1.1. 自動車システムのソフトウェア更新における特徴

ソフトウェア更新の対象となる ECU の組み合わせは，電気自動車やガソリン自動車などの車種によっても異なる．このため，システム構成として 1 デバイスの場合が多いコンシューマエレクトロニクス分野の製品と比較すると，ソフトウェア更新対象は非常に多様である．

図 4-1 に示す通り，自動車システムのソフトウェア更新は静的・動的両面で多様である．すなわち，ガソリン車，電気自動車といった車種により ECU の構成が異なる．さらに，同じガソリン車でもモデルにより ECU 構成が異なる(静的多様性)．例えば，車種による違いに伴い，ガソリン車の更新はイグニッション OFF で，電気自動車の更新は充電中に，など更新条件が異なる可能性がある．また，同じ車種でも構成要素となる ECU ごとに，それぞれ更新条件や更新手順が異なる可能性がある．他にも，同じ車両でも，1 回目の更新と 2 回目の更新では更新対象となる ECU が異なる可能性もある(動的多様性)．例えば，1 回目の更新では単一の ECU を更新するが，2 回目の更新では複数の ECU を更新する必要がある，などのバリエーションが考えられる．

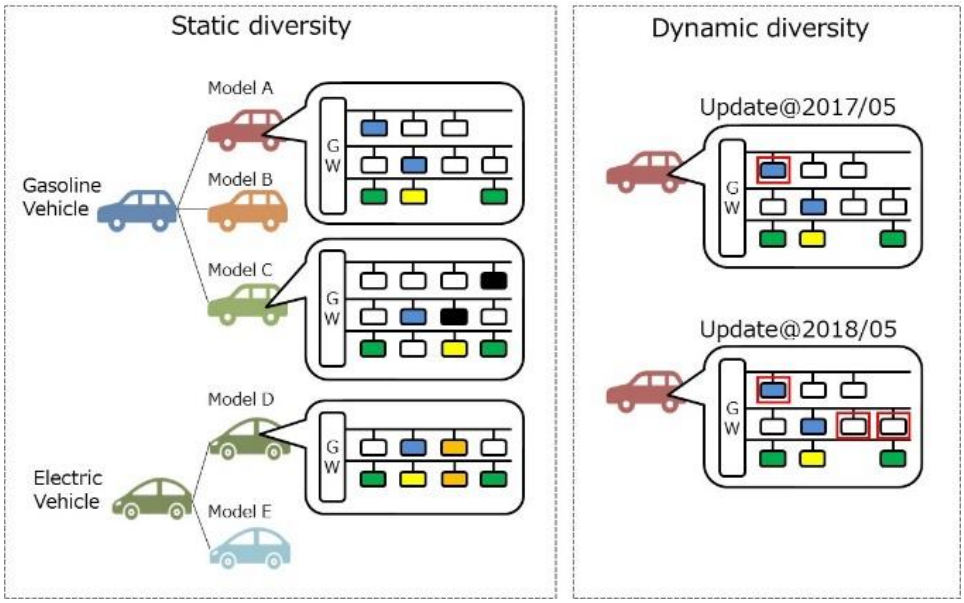


図 4-1 自動車システムの多様性.

4.1.2. 課題と目標

前述の通り，自動車のソフトウェアを更新する場合，従来はディーラで整備士が診断ツールを使ってマニュアルで実施していた．整備士の作業としては，例えば開始前の車両の状態チェック，更新対象 ECU と更新データの取得，実行中の車両状態の監視，完了後の結果確認などがある． 自動車システムに OTA を適用する場合，これらの更新手続きをシステムが自動で実行する必要がある．以下，このような更新手続きの実行を更新制御と呼ぶ．
本研究では，自動車システムに OTA を適用するための更新制御についての機能要件を整理し，これらの機能要件を満たしつつ，さらに自動車システム特有の多様性への対応が容易な OTA システムの機能配置と制御方式を提案する．

4.2. 自動車システム向け OTA 更新制御機能

図 4-2 に，本研究で提案する自動車向け OTA ソフトウェア更新システムの全体像を示す．本研究では，更新制御機能は車載ゲートウェイに配置し，そこで OTA センタから取得するパッケージに含まれる更新スクリプトに基づいて制御を実行する構成を提案する．更新スクリプトは，ISO 標準である OTX および ODX で記述されたシーケンスおよびパラメータをゲートウェイ上で実行可能な軽量な形式に変換されたものを用いる．ここで，OTX/ODX ファイルは自動車メーカや ECU の部品メーカにより作成され，OTA センタで軽量スクリプトに変換されて新プログラムそのものや差分データとともにパッケージに同

梱される構成を想定する．スクリプトの作成が自動車メーカと部品メーカで行われるのは，自動車全体や複数の ECU にまたがる部分は自動車メーカ，ECU 固有の部分は部品メーカが担当することを想定するためである．一方で，これらのスクリプト作成をすべて自動車メーカが行うことも考えられる．この場合も，スクリプトの入力元が変わるだけであり，本システムの構成に大きな影響はない．

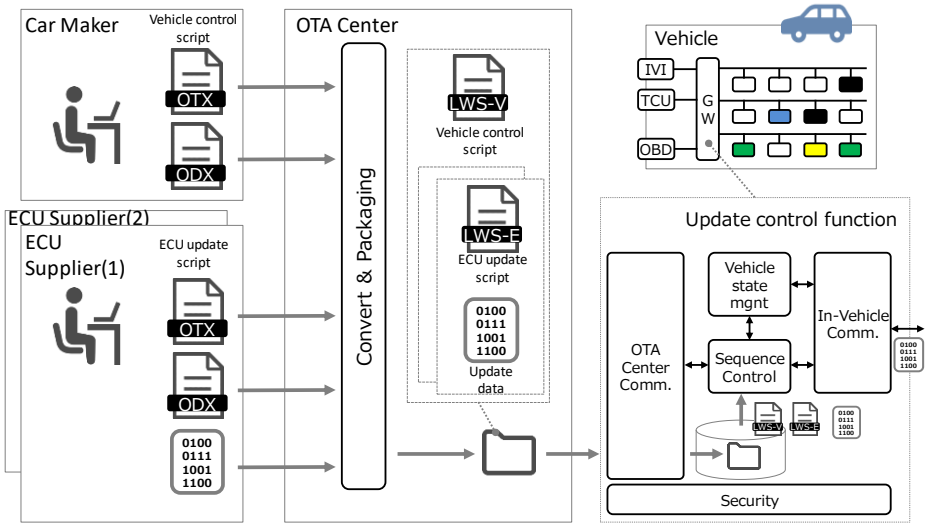


図 4-2 自動車向け OTA ソフトウェア更新システム全体像.

本章では最初に更新制御機能の要件を整理し，次に，要件に基づく機能配置についての検討結果を述べる．その後，自動車システム特有の課題といえる多様性に対応するための更新シーケンス制御の柔軟性確保の方法についての検討結果を述べる．

4.2.1. OTA 更新制御機能の要件

表 4-1 に更新制御機能への要件を示す.

表 4-1 更新制御機能の要件

Req#	内容
1	更新シーケンス制御が可能であること
1-1	シーケンス制御は多様なシステムに 対応可能な柔軟性を持つこと
2	車両状態の把握が可能であること
3	外部からの更新データの取得や外部との 構成情報の同期が可能であること
4	セキュリティ機能を備えること

(1)更新シーケンス制御

更新制御機能は，車両状態等に基づいてソフトウェア更新のプロセスの開始・中断や，Controller area network (CAN)などの車内ネットワークを介した ECU への更新開始指示，更新データ転送および ROM 書き換えなどの制御を行う必要がある．4.1.1 に述べたような多様性を持つ自動車システムに対して OTA を適用する場合，更新制御機能がガソリン車や電気自動車などの車種に起因する手順の相違や，ECU の違いに起因する手順の相違に柔軟に対応可能であること(Req#1-1)が特に重要となる．本課題については，0 にて詳細要件の整理を行う.

(2)車両状態の把握

更新制御機能は，前述のシーケンス制御を行うため，また，車両状態を OTA センタと同期するため，車両状態を把握できる必要がある．ここでの車両状態には，イグニッション OFF/ON，電源状態，走行状態，ユーザとのインタラクション結果，有線/OTA でのソフトウェア更新状態などが含まれる.

(3)更新データの取得および構成情報の同期

更新制御機能は，更新対象 ECU に適用する新プログラムや付加情報など更新用のデータ（以下，パッケージと称す）を外部から取得する必要がある．また，必要に応じて取得したデータを蓄積する必要がある．他にも，更新の要否を判断するため，ECU のソフトウェアバージョン情報などの構成情報を外部とやり取りする必要がある.

(4)セキュリティ機能

外部からの攻撃等により更新制御機能ののっとりや不正動作が引き起こされると，不正なプログラムの書き込みにより ECU の不正動作や機能停止が引き起こされる可能性がある．これらは自動車システムの安全性に係るため，更新制御機能は，OTA センタとの通信等のセキュリティを確保し，取得した更新データの正当性を保証する必要がある．

4.2.2. OTA 更新制御機能の配置

本研究では，前述の要件を満たす更新制御機能の配置先としてゲートウェイを提案する．これは，表 4-3 の機能配置に関する比較結果に基づく．比較の観点と比較観点とした理由は以下の通りである．

(A).ECU へのアクセス性

車両状態の把握および更新シーケンス制御のため，車両内の ECU へのアクセス性が容易であることが必要となるため．本評価では，更新制御機能から ECU までの経路となるコンポーネント数に基づき比較する．

(B).セキュリティ

更新制御に当たっては，セキュリティ確保が必要であるため．本評価では，JASO TP-15002[141]に規定された脅威分析を行い，Common Vulnerability Scoring System (CVSS)[142]に基づくリスク評価手法である CVSS based Risk Scoring System (CRSS)方式を用いて抽出したリスクの基準値 (CVSS 基本値)の最悪値に基づいた比較を行う．ここで，基本評価値のパラメータは，JASO TP-15002 に記載の値を用いる．また，保護資産を「OTA 更新制御機能」および「パッケージ」の 2 つとする．

(C).OTA センタへのアクセス性

OTA センタからのパッケージの取得，車両状態の OTA センタとの同期のためには OTA センタへのアクセス性が容易であることが必要となるため．本評価では，更新制御機能から OTA センタまでの経路となるコンポーネント数に基づき比較する．
経路となるコンポーネント数のカウントおよびセキュリティリスクの評価に用いたモデルを図 4-3 に示す．

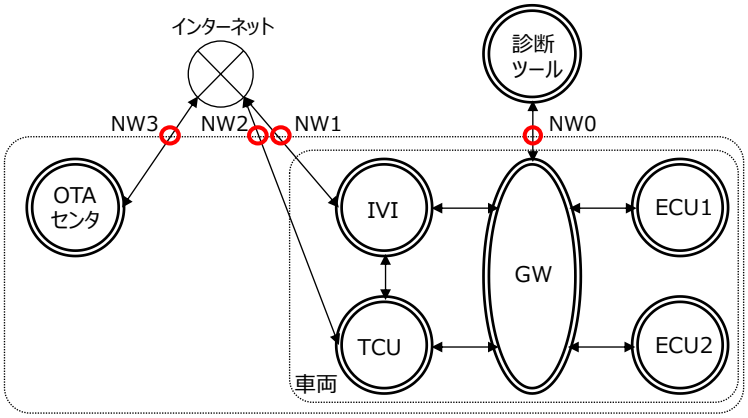


図 4-3 OTA システムモデル.

(D).ハードウェアリソース（CPU，メモリ）

更新制御はソフトウェアで実現することを想定した場合，更新制御機能が搭載可能なリソースを持つことが必要である．一例として，関連研究[33]の提案手法を用いて更新制御機能を実現する場合，数 MB の RAM，ROM が必要になると考えられる．本稿にて想定する車載デバイスのリソースを表 4-2 に示す.

表 4-2 車載デバイスの想定リソース

車載デバイス	RAM	ROM	CPU
IVI	数 GB	数十 GB	数 GHz
TCU	数百 MB	数百 MB	~GHz
Gateway	数百 KB	数 MB	数百 MHz~GHz
ECU*	数十 KB~数 GB	数百 MB~数 MB	

*ECU は用途により多様な構成が考えられる.

表 4-3 更新制御機能の配置先比較

車載デバイス	(A)	(B)	(C)	(D)
OTA センタ	3	8.5	0	大
IVI	2	8.5	1	中
TCU	2	8.5	1	中
Gateway	1	7.3	2	小
ECU*	0	6.4	3	小

更新制御機能を OTA センタに配置する場合，ECU までの経路上のコンポーネント数は 3 と最も多く，ECU へのアクセス性が悪いと言える．特に，OTA センタと車両の間の通信の安定性が保証されていない場合，更新制御中に ECU へのアクセスが遮断されて更新が継

続できなくなるリスクがある．OTA センタに配置した場合の CVSS 基本値の最悪値は 8.5 であり，インターネット(NW3)から更新制御機能に不正な動作を引き起こす攻撃のリスクが最も高い値となった．一方で，パッケージは OTA センタで生成・管理されるため，経路上のコンポーネント数は 0 と最も少なく，データ取得は容易である．また，組込み端末であるそのほかの配置先と比較した場合，十分なリソースがあると言える．

更新制御機能を IVI に配置する場合, ECU までの経路上のコンポーネント数は 2 であり，ECU へのアクセスは OTA センタと比較して良好といえる．本配置における CVSS 基本値の最悪値は 8.5 であり, インターネット(NW1)から更新制御機能に不正な動作を引き起こす攻撃のリスクが最も高い値となった．本モデルでは，IVI は携帯電話を用いたテザリング等で外部ネットワークに接続されており，OTA センタにアクセスするための経路上のコンポーネント数は 1 であり，データ取得は容易であると言える．また，地図やユーザインタフェースを備える IVI はそのほかの配置先と比較した場合，OTA センタほどではないがリソースには余裕があると言える．

更新制御機能を TCU に配置する場合，ECU や OTA センタへのアクセス，セキュリティリスクは IVI と同等である．また，近年の LTE 等高速回線に接続する機能をもつ場合，そのほかの配置先と比較した場合，OTA センタや IVI ほどではないが，リソースは多いと言える．

更新制御機能をゲートウェイに配置する場合，ECU までの経路上のコンポーネント数は 1 であり，ECU へのアクセスは非常に良好といえる．本配置における CVSS 基本値の最悪値は 7.3 であり，IVI 等を経由して遠隔(NW1 など)から更新制御機能に不正な動作を引き起こす攻撃のリスクが最も高い値となった．すなわち，IVI 等のように直接インターネットに接続し，遠隔での攻撃を直接受ける可能性が少ないため本配置のほうがセキュリティ上優位であると考えられる．また，元々の外部からのセキュリティ保護のためのコンポーネントとしての位置づけから，ハードウェアセキュリティモジュールなどを備え，攻撃への対策が行いやすいと言える．一方で，外部ネットワークとの接続は TCU や IVI を介して行う必要があるため，OTA センタまでの経路上のコンポーネント数は 2 となり，データ取得は少し困難となる．また，現在の車載ゲートウェイ用マイコンのリソースは TCU ほど潤沢ではない場合が多い．

更新制御機能を ECU に配置する場合，経路上のコンポーネント数は 0 である．ゲートウェイで保護された領域に配置することになるため，セキュリティ保護は良好であり，CVSS 基本値の最悪値は 6.4 である．外部ネットワークへの接続はゲートウェイおよび TCU または IVI を経由する必要があり，経路上のコンポーネント数は 3 となるため，OTA センタへの接続性は悪い．また，大半の ECU では，本来の制御機能以外を動作させるためのリソー

スの余裕がない場合が多いと考えられる．さらに，更新対象となる ECU それぞれに更新制御機能を搭載することは機能重複が多くなりコストアップにつながる．

以上の比較結果から，配置先として優位であるのは IVI,TCU,ゲートウェイであると考えられる．本研究では，低価格帯の車両など IVI が搭載されない車両があり得ること，ソフトウェア更新に当たってはセキュリティが重要であり，そのためのセキュリティを確保しやすい点を考慮し，ゲートウェイを更新制御機能の配置先として提案する．ゲートウェイにはリソースが少ないという課題があるため，少ないリソースで実現可能な更新制御方式を検討する．

4.2.3. OTA 更新制御の柔軟性確保

本項では，4.2.1 節の(1)において述べた更新シーケンス制御機能の柔軟性に関する要件(Req#1-1)が，さらにどの様な要件から構成されるかを詳細化し，これらを満足する更新制御方式の検討を行う．

(1-1-1)　処理実行有無の制御
各シーケンスにおいて，ECU の異常状態を示す Diagnostic trouble code(DTC)の確認をする／しない，ダウンロードしたソフトウェアの適用可否についてユーザの許諾確認をする／しないなどの処理実行有無の制御が必要となる．

(1-1-2)　状態判定閾値の変更
ソフト更新可否判断のための電池残量などの判定条件の変更等が可能である必要がある．

(1-1-3)　処理順序の入れ替え
複数の ECU を更新する場合の順番の制御など，シーケンス内の処理順序の変更が可能である必要がある．

(1-1-4)　新規処理の追加
状態確認対象 ECU の追加やその手順の追加が可能である必要がある．
（例：ガソリン車：イグニッション状態の確認．
電気自動車：イグニッション状態の確認＋充電状態の確認）

(1-1-5)　省リソース
車載ゲートウェイに搭載するため，省リソースで動作する必要がある．

(1-1-6)　運用容易性

更新制御機能において様々なシーケンスの実行を実現することは，当該シーケンスの生成・管理が必要になり運用が複雑になる可能性があるため，運用が容易である必要がある．
多様性に対応するために，それぞれの車種・モデルごとに更新制御ソフトウェアを開発する方法が考えられるが，それには開発コストの上昇という問題がある．

そこで，このような問題に対応するため，前述の要件を満たす方法として，以下の 2 通りの制御方式を比較した．2.2.1 節に述べた通り，制御内容を柔軟に変更するためのアプローチとしては，制御ソフトウェアを構成するバイナリモジュールの更新や，ファームウェアそのものを更新する方式も考えられるが，メモリリソースや更新時間，運用コストを考慮してパラメータによる制御及びスクリプトによる制御を比較対象とする．

(a)　　パラメータによる制御方式

OTA センタからダウンロードするパッケージに，更新制御に関するパラメータを設定し，更新制御ソフトウェアはパラメータに基づいて制御を行う．

(b)　　スクリプトによる制御方式

OTA センタからダウンロードするパッケージに，シーケンスを記述したスクリプトを同梱し，更新制御ソフトウェアは当該スクリプトを実行して制御を行う．

表 4-4 に，両方式の比較結果を示す．

表 4-4 OTA 更新のシーケンス制御における要件

Req#	内容	(a)	(b)
1-1-1	処理実行状態有無の制御	○	○
1-1-2	状態判定閾値の変更	○	○
1-1-3	処理順序の入れ替え	×	○
1-1-4	新規処理の追加	×	○
1-1-5	省リソース	○	×
1-1-6	運用容易性	○	×

パラメータ方式は，リソースの消費が少ないが，順番の入れ替えや新規処理の追加など要件を満たせない場合がある．また，同一車両において更新キャンペーンごとに異なる可能性のあるパターンも網羅してパラメータ設計を行うことは困難である．一方で，限られた範囲での柔軟性となるため，パラメータシートを作成して管理することができるなど，運用は容易であると言える．

一方，スクリプトを用いる方式は，Req#1-1-1～1-1-4 を満たすが，小リソースな機器上での実行は困難な可能性がある．また，柔軟性が高く処理内容を自由に定義できるため，作成や管理への負担が大きくなる可能性があり，運用が困難であると考えられる．
本研究では柔軟性に対する要件の満足性が高いと考えられるスクリプト方式について，運用が容易かつ小リソースな機器上で実行可能となる更新性制御方式を検討する．

4.2.4. ISO 標準技術によるスクリプト記述

標準技術を適用することで，自動車メーカ同士または自動車メーカと部品サプライヤ間でのスクリプトの共有が容易になる．

また，これらをオーサリングする市販ツール[143]があるため，記述に当たって市販ツールを利用することで，市販ツールを利用して従来運用されている手順と同様の運用を行うことが可能となる．

一方で，これら標準は XML での記述を前提としているため，リソースの少ない車載ゲートウェイ上で解析・実行することは困難である．そこで，本研究では ISO 標準で作成されたスクリプトを，OTA センタで軽量スクリプトに変換して車載ゲートウェイ上で実行する方式を提案する．

4.2.5. 軽量スクリプト言語を用いた車載ゲートウェイアーキテクチャ

図 4-4 に，車載ゲートウェイ上の更新制御ソフトウェアのアーキテクチャ概略を示す．

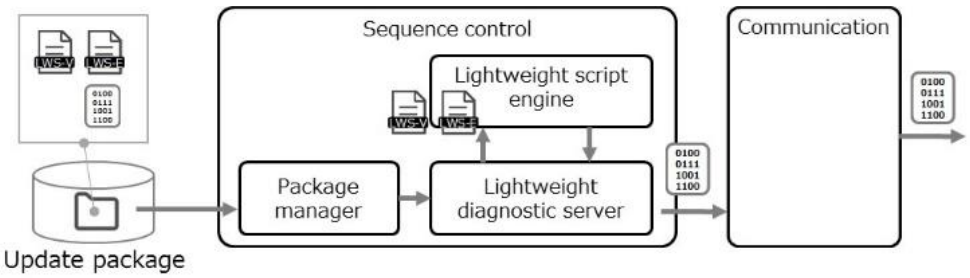


図 4-4 更新制御ソフトウェアアーキテクチャ概略．

Package manager は，更新パッケージを解析し，スクリプトや ECU への送信データを分離する．

Lightweight diagnostic server は，更新パッケージに含まれるスクリプトを読み出し，

Lightweight script engine を呼び出す．また，軽量スクリプトに ECU への命令を抽象化した API を提供し，軽量スクリプトからの呼び出しに応じて UDS コマンドの生成や受信した応答の解釈を行い，結果をスクリプトに伝える．さらに，更新パッケージに含まれる ECU への送信データを取得し，軽量スクリプトからの指示に従って ECU に送信する．
Lightweight script engine は軽量スクリプトの実行環境であり，OTA センタから受信したスクリプトを実行する．

このように UDS コマンドの構築・解析といったバイナリデータ処理など，負荷が高い処理は C 言語で実装した Lightweight diagnostic server に配置し，比較的処理負荷が小さいシーケンス制御などをスクリプト側の処理とすることで，柔軟性と処理負荷の低減を両立できる．

4.3. 評価

更新制御機能における要件の実現性を検証するため，提案方式の評価を行った．表 4-5 に機能要件とそれに対する評価内容を示す．本研究では，センタとの接続機能については，評価の対象外とし，ECU とのインタフェースを持つ機能を中心に，更新制御機能による車載システム制御の実現性を評価した．スクリプトとしては，組込みスクリプト言語の中でも軽量な Lua[68]を利用した．また，セキュリティアルゴリズムとしては，パッケージの情報秘匿に用いる共通鍵暗号，認証および改ざん検知に用いる公開鍵暗号，ハッシュアルゴリズムとしてそれぞれ AES(CTR モード，鍵長 128bit)，RSA(鍵長 3072bit)，SHA256 を評価した．評価にあたっては，OpenSSL[144]をベースに実装した．

表 4-5 更新制御機能の要件と評価内容

Req#	内容	評価内容
1 1-1	更新シーケンス制御	(a)スクリプトでの更新シーケンス実行 に必要なメモリ使用量 (b)スクリプトでの ECU 更新時間 (c)スクリプト変換処理の実現性
2	更新状態の把握	(a)パッケージを保護するセキュリティアルゴリズムに必要なメモリ使用量
3	更新データ取得と構成同期	本研究の研究対象外とする．
4	セキュリティ	(a)パッケージを保護するセキュリティ アルゴリズムに必要なメモリ使用量

4.3.1. 評価環境

図 4-5 に評価環境の外観を示す．提案方式の車載ゲートウェイ搭載可否を評価するため，車載ゲートウェイ向けのマイコンに前述のアーキテクチャのソフトウェアを実装した．車載ゲートウェイ向けマイコンとしてルネサスエレクトロニクス社製の RH850/F1L[145]を用い，これをテセラ・テクノロジー社製のマイコン評価ボード FL-850/F1L-176-S[146]に搭載して評価環境を構築した．また，更新対象となる ECU は同じ車評価環境を利用し，車両状態を模擬するアプリケーションは PC 上のアプリとして実装して評価した．評価用ゲートウェイと PC は，Vector 社製の VN1610[147]と XL ドライバライブラリ[148]を用いて CAN で接続した．CAN の通信帯域は 500Kbps を利用した．

表 4-6 に評価環境の詳細を示す．

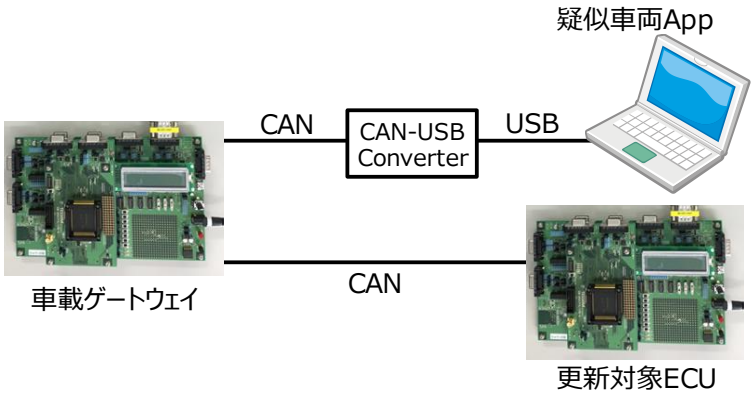


図 4-5 評価環境外観

表 4-6 評価環境

項目		値と説明		
Gateway and Target ECU	CPU	80MHz Renesas RH850/F1L		
	ROM	2MB		
	RAM	160KB		
Pseudo vehicle application	CPU	Intel(R) Core(TM) i5-6500 CPU @ 3.2GHz		
	RAM	4.00GB		
	OS	Windows 7		
CAN	Bandwidth	500Kbps HI-Speed CAN 帯域 100%利用		
	Protocol	ISO14229(UDS), ISO15765-2		
	Parameter	Separation time=0, Block size=1		
Lightweight script engine		Lua5.3 (virtual machine with standard libraries)		
Update data		走行系 ECU 制御ソフトウェアの 差分データ(45,178byte)		
Security algorithm		秘匿	共通鍵暗号	AES128
		認証および 改竄検知	公開鍵暗号	RSA3072
			ハッシュ	SHA256

評価用車載ゲートウェイ，更新対象 ECU および疑似車両アプリの機能配置を図 4-6 に示す．車載ゲートウェイにはスクリプトで規定された手順に従い更新シーケンスの制御を行う更新シーケンス制御部，ISO15765-2[50]に対応した CAN 通信を行う CAN ミドル及び CAN ドライバを搭載する．更新制御部は，提案の通り Package manager，Lightweight diagnostic server および Lightweight script engine で構成する．本研究では，パッケージはダウンロードされた状態を想定してあらかじめ ROM に配置し，これを読み出す．Lightweight diagnostic server を AUTOSAR OS 上の 1 タスクとして実装し，これがライブラリとしての Lightweight script engine と Package Manager を呼び出す構成とした．

Lightweight script engine には，script から呼び出される API として，バージョン情報取得や ECU のモード変更，更新データ転送，ソフトウェア更新の結果確認要求などを実装する．ここで更新データを転送する API は，UDS に規定されたデータ転送のサービス RequestDownload，TransferData，RequestTransferExit の順次発行するように実装し，script に記述が必要となるステップを削減するようにした．更新制御時の評価にあたっては，Lightweight diagnostic server は外部からのトリガによって Package manager を呼び出し，実行するスクリプトを抽出する．次に，Lightweigth srirpt engine を呼び出して抽出したスクリプトを実行し，車両状態を把握するシーケンスや ECU 更新のシーケンスを実行する．本研究では，車両状態の把握のための Vehicle state manager を Lightweight diagnostic server 内の 1 機能として実装し，取得した車両情報の管理を行う．また，セキュリティアルゴリズムは Package manager の一部として実装した．ただし，パッケージの検証と復号は ECU 更新とは別にダウンロード時などに行われることを想定し，ECU 更新時の処理には含めない構成とした．

更新対象 ECU には，同様の通信ミドルウェアおよび車載ゲートウェイからの指示に従って更新を実行する機能を搭載する．更新対象 ECU 内の UDS server は車載ゲートウェイから発行されるコマンドの解析を行い，Update logic は差分の復元や Flash ROM の消去・書き込みなどの制御を行う．

疑似車両アプリは，Windows PC 上のネイティブアプリケーションとして実装した．評価用車載ゲートウェイとの接続に必要な CAN 通信機能は，Vector 社の提供する XL Driver Library を利用した．

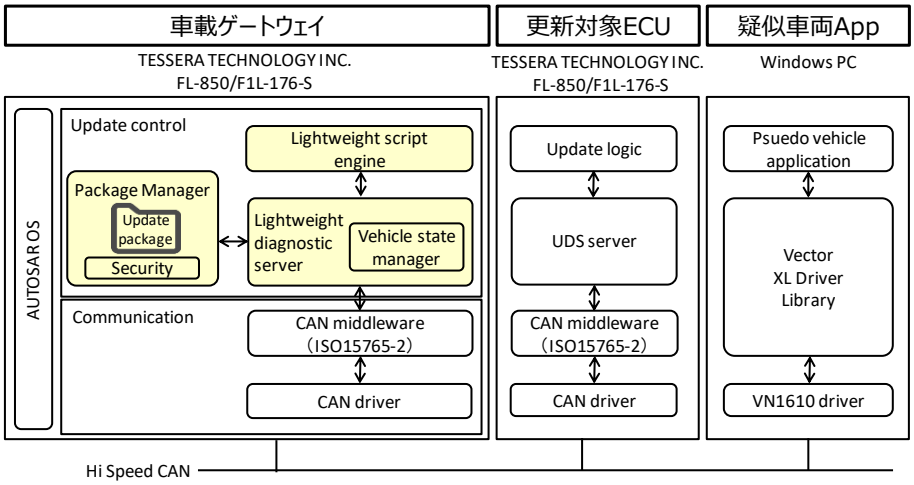


図 4-6 評価環境機能配置

車両状態把握機能については，図 4-7 に示すシーケンスを評価した．車両システムの車両状態の把握は，周期メッセージなどで CAN バス上を流れている情報を取得する方法と

ECU に問い合わせて取得する方法の 2 通りがある．本研究では，前者の情報として車速，シフトポジション，ブレーキ状態，後者の情報としてバッテリの残量を想定し，これら 4 つの情報を更新開始可否を判定する場合を想定した評価とした．この更新開始条件も多様であるため，状態確認のシーケンスもスクリプトで実現することが好ましい．また，開始条件の判定にあたっては，車両内を流れるすべての情報を管理する必要はないため，本研究では，取得する情報をスクリプトから設定する構成を評価した．

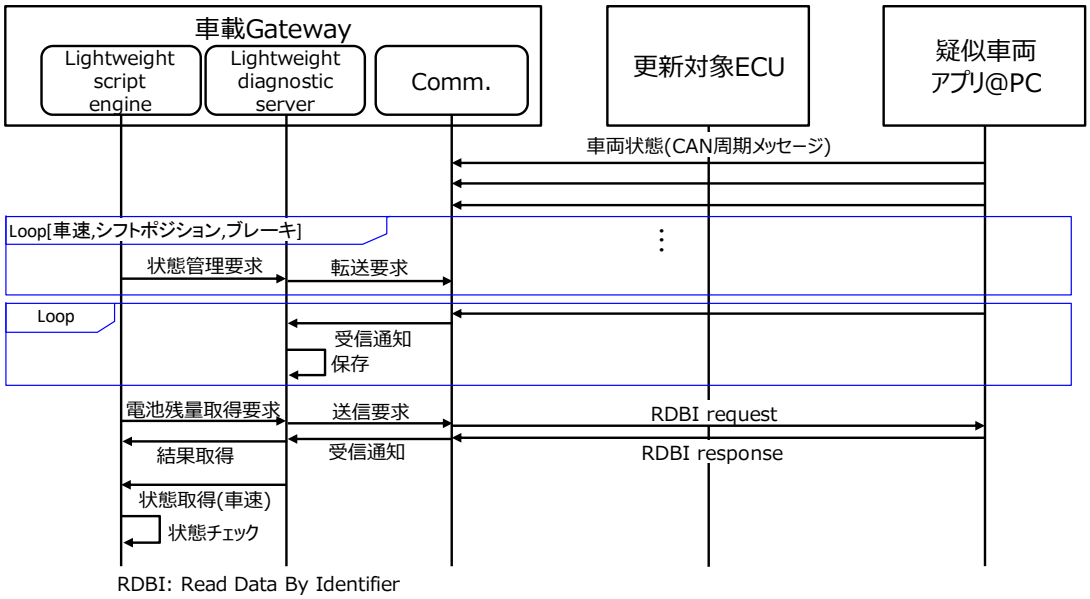
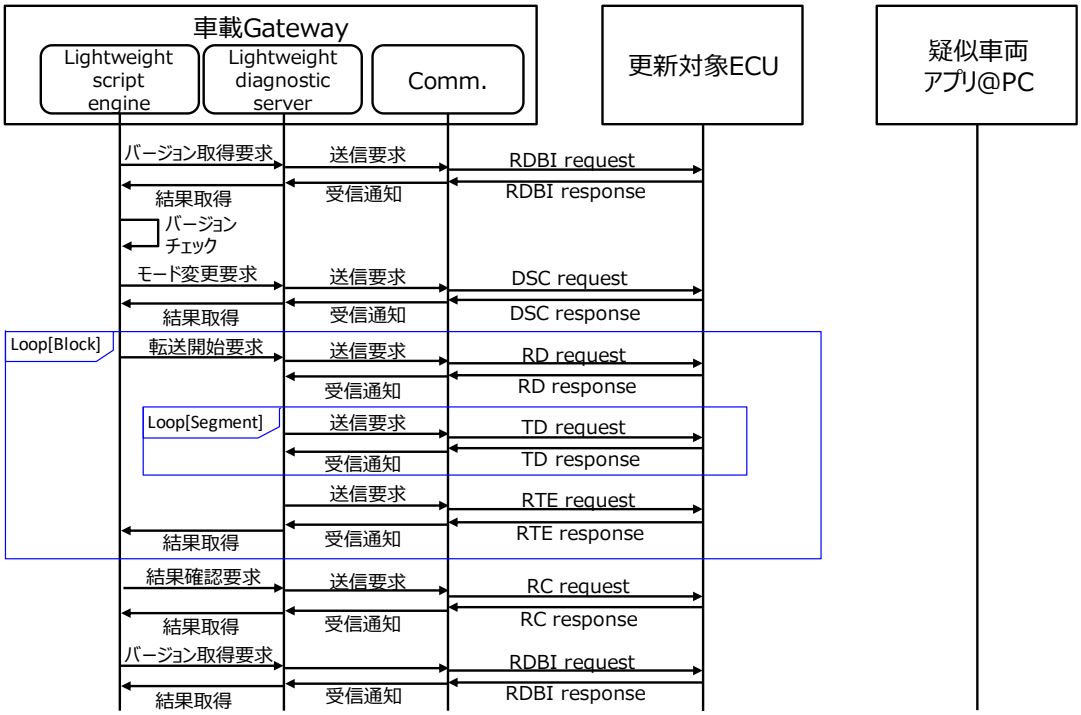


図 4-7 車両状態管理機能評価用更新シーケンス

更新シーケンス制御機能の評価には図 4-8 に示す簡易的な更新シーケンスを用いた．本評価の簡易シーケンスでは，ECU のソフトウェアバージョンの取得と確認，ECU に対するプログラム書き換えモードへの変更，更新プログラムの転送と ROM 書き込み，書き込み後の新プログラムの検証を行う．更新前のバージョンチェックは，部品交換や整備工場での有線ソフトウェア更新などにより ECU のソフトウェアバージョンが OTA センタと同期されているものと異なる場合の不具合を防止するために実施する．例えば，差分更新を行う場合には，配信された差分データの基バージョンと ECU のバージョンが異なる場合は正常なデータの復元ができない．基本シーケンス以外にも，実運用で想定されるシーケンスとして，故障状態の取得，更新前の ECU のデータ書き込みを有効にするためのセキュリティロックの解除，パラメータの再設定など ECU 個別の処理が実行される．また，更新対象の ECU が他の ECU と依存関係を持つ場合には，その ECU のバージョンチェックも実行する必要がある．車両に搭載される ECU のバージョンチェックは，更新直前のみでなく，OTA センタとの構成情報の同期のためにシステム起動時に毎回実施するのが望ましい．



RDBI: Read Data By Identifier, DSC: Diagnostic Session Control, RD: Request Download
TD: Transfer Data, RTE: Request Transfer Exit, RC: Routine Control

図 4-8 ECU 更新制御評価用更新シーケンス

以上の評価環境により評価する内容とその方法を表 4-7 にまとめる．メモリ使用量及び ECU ソフトウェア更新時間は，提案内容の車載ゲートウェイへの搭載可否の検証のために評価する．また，スクリプト変換処理は，配信にあたっての運用負荷低減のために，OTA センタにおいて機械的に実行されることを想定する OTX から Lua スクリプトへの変換処理の自動化可否の検証のために評価する．

表 4-7 評価項目と方法.

評価内容			評価方法
メモリ使用量	ECU 更新制御	Req#1(a)	前述の評価環境で Lua によるメモリ確保要求を積算
	状態管理	Req#2(a)	
	セキュリティ	Req#4(a)	メモリマップより算出
ECU ソフトウェア更新時間		Req#1(b)	前述の評価環境で実測
スクリプト変換処理		Req#1(c)	OTX から Lua スクリプトへの変換における課題評価

4.3.2. 評価

(1)メモリ使用量

評価環境上で前述のシーケンスを実行した際の車載ゲートウェイにおける Lightweight script engine の RAM および ROM の使用量を表 4-8 に示す．また，比較対象として，関連研究[33]で用いられた OSGi フレームワークを用いた場合の想定メモリ使用量を併記する．OSGi フレームワークの使用量は(株)日立ソリューションズ社製品の公表値[149]を用いた．また，Java VM の使用量は Oracle 社の公表値[150]を用いた．

表 4-8 提案方式によるメモリ使用量[KB]

Item	提案方式(Lua VM)		OSGi with JavaME	
	状態取得	ECU 更新	OSGi	Java VM
RAM	15.6	12.3	6,000	128
プログラム(ROM)	76.0		100	1,000

表 4-9 に前述のセキュリティアルゴリズムを車載ゲートウェイ用マイコンに搭載する場合のメモリ使用量を示す.

表 4-9 セキュリティアルゴリズムのメモリ使用量[KB]

Item	秘匿機能	認証/改ざん検知機能	
	AES128	RSA3072	SHA256
RAM	0.4	11.6	0.2
プログラム(ROM)	13.0	129.8	4.6

本評価により，セキュリティアルゴリズムを含む更新制御機能で使用するメモリが RAM は 16KB 程度，ROM が 224KB 程度とメモリ使用量の観点からは提案方式が車載ゲートウェイ用マイコンに搭載可能であることが確認できた.

(2)更新制御機能処理時間

評価環境上で前述のシーケンスを実行した際の ECU ソフトウェア更新時間とその内訳を表 4-10 に示す.

表 4-10 ECU ソフトウェア更新時間と内訳.

Item		Time[ms]	Ratio[%]
更新時間		14,392	100.0
	通信&ECU 処理	14,074	97.8
	更新制御機能処理	318	2.2

本評価における更新制御機能の処理時間は全体の 2.2%であり，スクリプトを用いることにより処理時間に与える影響は軽微であることが確認できた.

(3)スクリプト変換処理

OTX によるシーケンス記述を Lua スクリプトに変換するに当たっては，表 4-11 に示すステートメントについて対応が困難であることが判明した．これらの課題は前述のシーケンス，すなわち，ECU をひとつずつ順次更新するケースを実現するに当たっては問題とならない.

表 4-11 Lua に変換困難な OTX ステートメント.

#	ステートメント	内容
i	Parallel	並列処理
ii	Exception	例外処理
iii	break	loop 指定してのループ停止

(i)Parallel

OTX では Java で規定されているような thread による並列処理の記述が可能である．一方，Lua の並列処理はコルーチンで実装されており，スレッドの切り替えを明示的に記述する必要がある．並列処理を行うには，OTX にスレッド切り替えを行うためのステートメントを追加する，あるいは Lua 側で疑似的にマルチスレッドを実現するような記述への変換を行う必要がある.

(ii)Exception

OTX では Java で規定されているような try-catch による例外処理が記述できるが，Lua には相当する要素が存在しない．そのため，OTX スクリプト作成時に Exception を用いない制限を行うか，Lua 側に例外処理機構を実装する必要がある.

(iii)Break

OTX の break ステートメントでは多重ループの場合停止するループの指定が可能であるが，Lua ではそのような機能はない．そのため，OTX スクリプト作成時に脱出するループを特定してループを抜ける break を用いない制限を行うか，Lua 側にループを指定して処理を停止できる機構を実装する必要がある.

4.4. 考察

本章では，自動車システムのソフトウェアを遠隔で更新するための更新制御機能について，機能要件を整理し，自動車システムの構成と特徴に基づいて機能配置と制御方式を提案した．また，車載ゲートウェイ向けマイコンへの実装評価によって，OTA センタとの接続機能を除く提案手法が実現可能であること示した．ただし，以下に述べる課題については，引き続き検討が必要である．

更新シーケンス制御機能については，単一 ECU の更新についての評価を行った．本評価の結果は，依存関係のない複数の ECU を逐次更新する場合には同様に適用可能である．一方で，実運用において，依存関係のある複数の ECU を同時に更新する場合の検討がさらに必要である．このような場合には，1 つのパッケージにこれらの更新データとスクリプトを同梱する．更新用のパッケージを格納する領域の制限等から，例えば数十個の ECU を同時に更新することは運用上困難であり，同時に更新が行われる ECU の数はせいぜい数個である．しかし，これらは，依存関係を持つため，依存関係を考慮した更新順序の制御や，更新失敗時の制御が必要となる．例えば，1 つの ECU の更新に失敗した場合は，書き換えが完了した他方の ECU を元に戻すような制御が考えられる．また，複数の ECU の更新を短時間で行うためには，更新の並列化が有効である一方で，並列化を行う場合は RAM 使用量が増加する可能性がある．この場合は，コルーチンによる並列化を実装することで，RAM 使用量を大きく増加させることなく，複数 ECU の同時更新の時間短縮が可能であると考えられるが，前述の通りスクリプト変換処理に課題がある．

セキュリティ機能については，暗号化やデジタル署名によるパッケージの保護を想定し，これらに対応するためのセキュリティアルゴリズムの車載ゲートウェイマイコンへの搭載可否をメモリ使用量の観点で評価した．一方で，セキュリティの実運用のためには，アルゴリズムの搭載性に加え鍵情報などの管理が重要である．ゲートウェイは自動車システムのセキュリティ保護のために搭載されるため，ハードウェアセキュリティモジュールなどが搭載される．そのため，前述のセキュリティ保護に必要な鍵情報などをセキュアに管理することが可能であり，これらの保護機能を搭載することは容易であると考えられる．また，TLS などによる OTA センタとの通信路の保護も必要となるが，OTA センタとの通信路の保護については，後述の OTA センタとの通信機能と合わせて検討が必要である．

表 4-1 に挙げた要件を満足するためには，さらに更新データの取得および構成情報の同期機能，すなわち OTA センタとの通信機能の検討が必要となる．本機能については，車種等に依存する部分はほとんどないと考えられるため，スクリプト等での対応による柔軟性の確保は必要ない．関連文献のように OMA DM のクライアント機能を搭載することで実現

が可能となる．ただし，ゲートウェイはリソースが少ないため，最小限の機能を搭載するなどの検討が必要と考える．

4.5. 結語

本章では，自動車システムにおける OTA によるソフトウェアの遠隔更新向けの更新制御方式を提案した．提案に当たり，更新制御機能への要件を整理し，当該要件に基づいて更新制御機能の配置先を提案した．さらに，従来 OTA が適用されてきたコンシューマシステムと異なる自動車システムの多様性という課題に対して，スクリプトを用いて更新シーケンスの制御を行う方式を提案し，これを実装してメモリ消費量，更新時間の観点から，提案方式が車載ゲートウェイ向けのマイコンに搭載可能であることを確認した．また，スクリプト変換の制限を明確にした．

以上により，更新手順が異なる可能性のある多様なノードで構成されるシステムにおいて，ソフトウェアの更新によりシステムを拡張するための制御機能の設計手法を示した．具体的には，制御機能への要求と前提とするシステム構成から制御機能の配置先を決定し，標準化された技術と軽量なスクリプトを組み合わせることで，セキュリティと運用性を向上させつつ，組込みシステム共通の課題であるリソース制約への対応が可能であることを示した．

2.4.1 節に述べた通り，ゲートウェイを含む車載 ECU 用のマイコンは高機能化し，利用可能なメモリリソースも増加している．さらに，将来 IoT 化が進展してゲートウェイがエッジコンピュータとしてのデータ集約等の処理を担うようになると，利用可能なリソースはさらに増大することが予測される．一方で，高機能化したゲートウェイは，ホームネットワークにおけるサービスゲートウェイと同様に，多数の機能を収容する必要があるため，それぞれの機能に割り当てが可能なリソースは制限される．また，自動車には自動運転機能を備えるような高級車以外にも，軽自動車のような低価格車の製品が存在する．さらに，車両は２輪車，や農機など，本章で論じた乗用車以外にも多様な分野で利用されている．このような低価格にシステムを構成する必要がある車両においては，将来にわたっても低コストで導入が可能な省リソースのゲートウェイが利用されるものと見込まれる．本提案の構成は，将来にわたっても有効であると考える．

第5章 リソース制約を考慮したソフトウェア差分更新方式

5.1. 緒言

2.2.2 節では，ネットワーク接続された組込みシステムのノードを再構成してシステム拡張を行う場合に，消費電力の抑制，可用性の確保などが課題であることを述べた．また，2.4.3 節では自動車システムにおいては，狭帯域な CAN を中心に構成される車載ネットワークを利用する ECU 更新時間の短縮と，組込みシステムの共通課題であるリソース制約について考慮した設計が課題であることを示した．

そこで本章では，CAN におけるデータ転送量を削減し更新時間を短縮する技術としての車載 ECU 向け差分更新方式の提案を通じて，リソース制約のあるノードに対して差分更新を適用するための手法について論じる．ベースとする差分更新方式としては，2.2.2 節で示した評価結果において最も圧縮効果の高かった bsdiff[87]を用いる．

5.2. 車載 ECU への差分更新適用課題

5.2.1. bsdiff の差分圧縮処理概要

bsdiff は，バイナリファイル間の差分を抽出する OSS である． 図 5-1 に，bsdiff による差分ファイルの生成手順及び生成される差分ファイルのフォーマットを示す．bsdiff では，最初に，Suffix Sorting を用いて生成した接尾辞配列を用いて新旧バイナリデータの一致部分を検索する．次に，当該一致部分を元に，近似一致部分と新規追加部分の分離を行う．近似一致部分とは，ソースコード上の変更は行われなかったが，参照アドレスの変更等により，プログラムのバイナリデータに変更が生じた部分に相当する．新規追加部分は，ソースコード上も新規に追加された部分に該当する．その後，隣接する近似一致部分と新規追加部分を 1 つの区画として，この区画に対する復元命令(Control)を生成する．抽出した近似一致部分については，新データと旧データの減算を行う．これにより，変更部分以外が 0 となる圧縮効率のよいデータ(DIFF)が生成される．復元命令は，DIFF 部のサイズ an，新規追加部分(EXTRA）のサイズ bn，及び復元時の旧プログラム上のポインタ移動量 cn から構成する（図 5-1 の①）．bsdiff では，これら各区画の Control，DIFF，EXTRA をそれぞれひとまとめにして bzip2[151]を用いて圧縮し，これにヘッダを追加して差分ファイルを生成している（図 5-1 の②）．

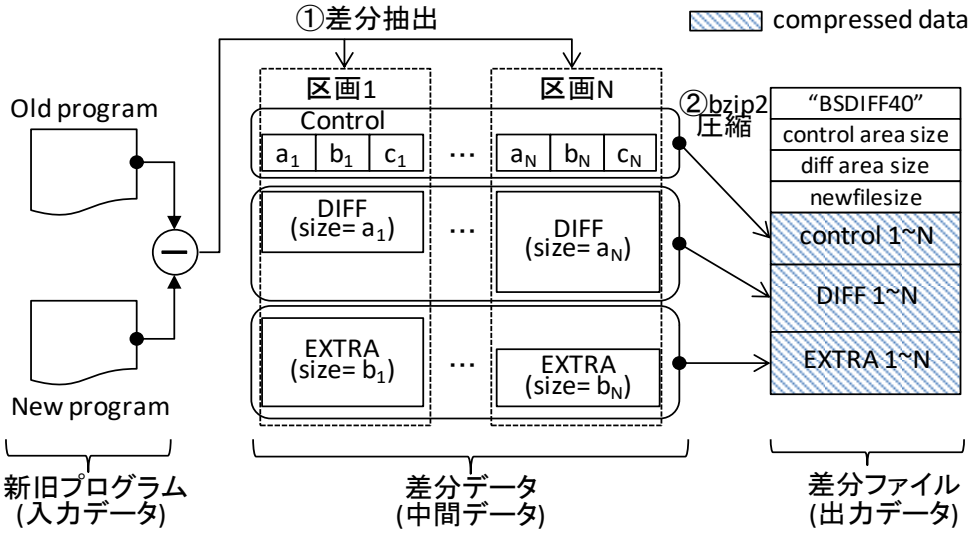


図 5-1 bsdiff の差分ファイル生成手順とデータ構成

5.2.2. bspatch における復元処理

bsdiff を用いて生成された差分ファイルは，bspatch と呼ばれるプログラムを用いて復元する．図 5-2 に bspatch による差分復元手順と復元時に使用するメモリ領域を示す．

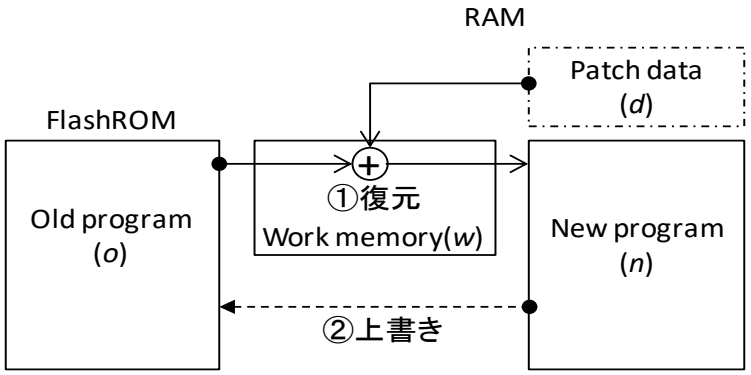


図 5-2 差分復元手順と必要メモリ

bspatch による差分復元では，Flash ROM に格納されている旧プログラムと，受信した差分ファイルから新プログラムを復元し，復元領域に格納する(①)．次に，Flash ROM 上の旧プログラムを復元した新プログラムで上書きする(②)．このため，復元に際しては式(1)で示すメモリ量が必要となる．

$$M = d + w + o + n \qquad (1)$$

ここで，dは差分ファイル格納領域のサイズ，wは復元に必要なワークメモリ，oは旧プログラムサイズ，nは新プログラムサイズである．以下，各領域の必要量について説明する．

(1)差分ファイル格納領域 d

bsdiff により出力される差分ファイルのサイズは，比較対象の類似度によって大きさが異なる．全く異なるデータを比較する場合等，十分に圧縮されない可能性もあり，復元側では保守的に領域を確保しておく必要がある．ECU のプログラムでは，メモリを動的に確保することはまれであり，基本的には用途に応じたメモリを静的に確保しておく．このため，差分ファイル格納領域としてプログラムサイズの 1/4 を確保すると仮定した場合 ，2MB のプログラムでは 500KB の領域が必要となる．

(2)ワークメモリ w

bsdiff では，前述の通り， Control 部，DIFF 部，EXTRA 部がそれぞれ別々に圧縮されており，復元時にこれらを並行して解凍する必要がある．このため，復元に際しては圧縮アルゴリズムが必要とする量の 3 倍のメモリが必要となる．bsdiff では，圧縮アルゴリズムとして bzip2 を利用しており，復元には 100kB+(2.5×ブロックサイズ)のメモリが必要である．ブロックサイズを 4KB に設定した場合でも，110kB のメモリが必要となるため，bspatch における復元では 330kB のメモリが必要となる．

(3)新旧データ格納領域 o , n

差分更新では，旧プログラムを参照しながら新プログラムを復元するため，旧プログラムの格納領域と復元した新プログラムの格納領域が必要となる．旧プログラムを Flash ROM から読み出すとして，復元用には，新プログラムと同じサイズの RAM が必要となる．

5.2.3. 適用課題と目標仕様

車載 ECU では，部品コストを抑制するため，OTA が先行導入されている携帯電話等と比較してメモリリソースが少ない．そのため，基本的にはプログラムコードは Flash ROM 上で実行される．また，プログラムを格納する Flash ROM のサイズと比較し，差分ファイル格納，新プログラム復元及びワーク領域として利用可能な RAM のサイズが小さい． 表 5-1 に主要な車載マイコンメーカの製品のメモリ構成を示す[120][152][153][154][155]．ここでは，RAM サイズ，実行するプログラムが書きこまれる Code Flash のサイズ，パラメータ等が書きこまれる Data Flash のサイズ及び Code Flash のブロック構成について示した．例えば，Renesas 社製のボディ系 ECU 向けマイコンである RH850/ F1L[145]では，

ROM サイズは 256KB～2MB，RAM サイズは 32KB～192KB である．このような特性から，差分更新を車載 ECU に適用するためには，復元処理における使用メモリ量を削減することが重要となる．

表 5-1 車載 ECU 用マイコンの仕様例

Vendor	Micro-controller	RAM [KB]	Code Flash [KB]	Data Flash [KB]	Code Flash-Block configuration
Renesas	RH850/E1L[4]	160	2048	64	8KB×8, 32KB×62
	RH850/F1L[5]	96 ~192	768 ~2048	32 /64	8KB×8, 32KB×22 8KB×8, 32KB×62
Infineon	AURIX TC277TP[8]	472	4096	384	不明
	AUDO TC1797[7]	224	4096	64	（16KB×8, 128KB×1, 256KB×7）×2
NXP	MPC5744P[9]	384	2560		16KB×4, 32KB×2, 64KB×6, 256KB×8
	MPC5741P[9]	128	1024		16KB×4, 32KB×2, 64KB×6, 256KB×2

差分更新の車載 ECU への適用のためには，RAM 使用量削減が重要となる一方，一般的に，メモリ使用量を削減した場合，圧縮率が下がる．そこで，本研究では，車載マイコンに搭載可能なレベルにメモリ使用量削減しつつ，bsdiff 相当のデータ削減率を実現し，ECU のソフトウェア更新の更新時間を短縮することを目標とする．

5.3. bsdiff の省メモリ化方式

bsdiff を用いる場合に，ECU における復元処理に必要なメモリ量を削減するための方式について，差分の取り方（差分生成単位），差分ファイル構造，圧縮アルゴリズムに着目して検討した結果を示す．

差分生成単位の検討では，広域ブロック差分方式により，差分サイズを抑えつつ復元した新データを一時格納する領域 n を削減する方式を提案する．差分ファイル構造の検討では，データを復元する順番に並べて圧縮し，逐次復元を可能とすることで差分ファイル格納領域 d とワークメモリ w を削減する方式を提案する．圧縮アルゴリズムの検討では，2 段階圧縮方式を適用してワークメモリ w を削減する方式を提案する．

5.3.1. 差分生成単位

Nakanishi らの提案[32]では，新プログラムを復元する前に必要な旧プログラムの一部を

削除してしまう Read-Write Conflict を，有向グラフを用いて回避しインプレース更新を適用することで新旧プログラム格納領域 *o,n* を削減する方式が示された．この方式では，前述の領域を大きく削減できる一方で，グラフを生成し格納するための領域がワークメモリ *w* に追加となる．また，プログラム全体を比較しているため，差分格納領域 *d* として十分な領域を準備しようとした場合, 例えばプログラムサイズの 1/4 などの大きな領域の確保が必要になる．本節では，前者の課題に対する方式提案を行う．後者の課題への対応については，次節で詳細を説明する．新旧プログラム格納領域 *o,n* を削減するため，本研究では Flash ROM の消去ブロックを基本単位として差分を生成するブロック差分方式を提案する．これにより，復元時に必要とする Work メモリの追加なしでインプレース更新を適用でき，復元した新プログラムを格納する領域 *n* を削減できる．消去ブロックを基本単位とした差分生成方法には，以下の 2 通りがある．

(1a) 新旧プログラムの同一アドレス範囲を比較（ブロック差分）
(1b) 新プログラムの 1 ブロックと，旧プログラムの未復元領域に相当する
　　複数ブロックを比較（広域ブロック差分）

図 5-3 に，各方式の概要を示す．
方式(1a)では，新旧プログラムの同一ブロックを比較し，差分を生成する．例えば，新プログラムの Block 1 は旧プログラムの Block 1 と比較され，Block 1 を復元するための差分ファイルが生成される．各ブロックについてこれを繰り返すことで，プログラム全体を復元するためのパッチファイルを生成する．

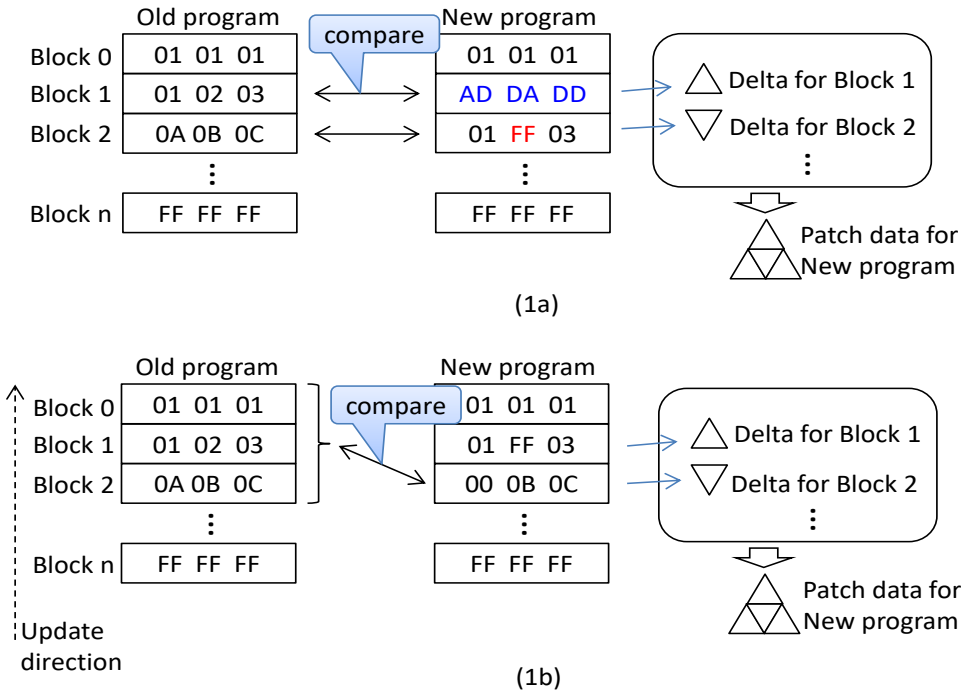


図 5-3 ブロック差分方式概要

方式(1b)では，新プログラムの 1 ブロックに対して，旧プログラムの複数ブロックを比較対象として差分を生成する. 例えば, 新プログラムの Block 2 については旧プログラムの Block 0,1,2 全体と比較し，Block 2 を復元するための差分ファイルが生成される．各ブロックについてこれを繰り返すことで，プログラム全体を復元するためのパッチファイルを生成する. 方式(1b)は, 新プログラムに大きな新規コードが追加される場合に有効であると考える. すなわち，図 5-4 に示すように新プログラムの Block 1 に相当する部分に新規コードが追加された場合，新旧プログラムの Block1 同士や Block2 同士を比較しても，一致する部分がほとんどなくなってしまう．これは，追加された部分以降の全ブロックに影響するため，結果として差分サイズが大きくなるという懸念がある．プログラムのコードが削除された場合も発生すると考えられるが，ECU のプログラムは PC や携帯電話のプログラムとは異なり，出荷後に機能削減やメモリマップの変更が行われることは少ないと考えられる．このため，本方式では，図 5-3 の(1b)に示すように，Block n から降順に差分生成を進め，旧プログラムの未復元領域全体を比較対象として利用する.

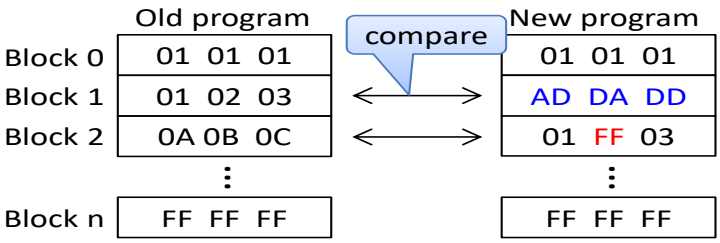


図 5-4 モジュールが挿入される事例

5.3.2. 差分ファイル構造

bzip2 を含む多くの圧縮アルゴリズムでは，ランダムな位置からの解凍はできず，データ先頭から，解凍位置や状態をワークメモリ領域に保持しながら解凍処理を行う必要がある．このため，図 5-1 の②に示した Control 部，DIFF 部，EXTRA 部をそれぞれひとまとめにして並列に圧縮する場合（並列圧縮）では，復元時に要するワークメモリ量は圧縮アルゴリズムが解凍に要するサイズの 3 倍となってしまう．データ構造の変更により本問題を解決する方式として，以下の 2 通りが考えられる．データ構造の変更によらずに復元時に利用するワークメモリを削減する方法としては，よりメモリ使用量の少ない圧縮アルゴリズムを採用する方法などがある．

(2a) 並列圧縮ではなく，差分データを一つに連結して圧縮（以下，直列圧縮と呼ぶ）し，復元時に必要な状態保持領域を削減する．

(2b) 復元データへのランダムアクセスが可能な圧縮アルゴリズムを利用し，並列に復元する際にもそれぞれの復元状態保持を不要にする．

ここで，(2b)については，一般化されたアルゴリズムが少ないことと，一般的な圧縮アルゴリズムでランダムアクセスを可能にする場合は，細かいブロック単位に分割する必要があるなどで圧縮効率が下がる可能性が高いことから，(2a)の方法を詳細検討することとした．

差分データを直列圧縮する場合のデータの直列化方式として，以下の 2 通りが考えられる．

(2a-1) 従来のデータ列のまま Control 部，DIFF 部，EXTRA 部をそれぞれまとめてから全データを 1 度に圧縮する

(2a-2) データを復元処理の順番に整列した後で圧縮する

図 5-5 にこれらのデータ構成の概略図，図 5-6 に差分復元手順の概略図を示す．圧縮率の

観点では，(2a-1)の方式では，類似の構造のデータが集まっているために圧縮率が良いと見込まれる．一方で，(2a-2)の方式は異なる構造のデータを整列しているため圧縮効率は良くない可能性がある．復元時のメモリ使用量の観点では，(2a-1)の方式では，差分ファイル全体を受信するまで復元ができない(図 9(2a-1))ため，差分ファイル格納領域 d が大きくなる．一方，(2a-2)の方式は，復元する順番にデータを受信し，逐次復元できる(図 9(2a-2))ので，差分ファイルを格納するために必要な領域は最小限に抑えられる．

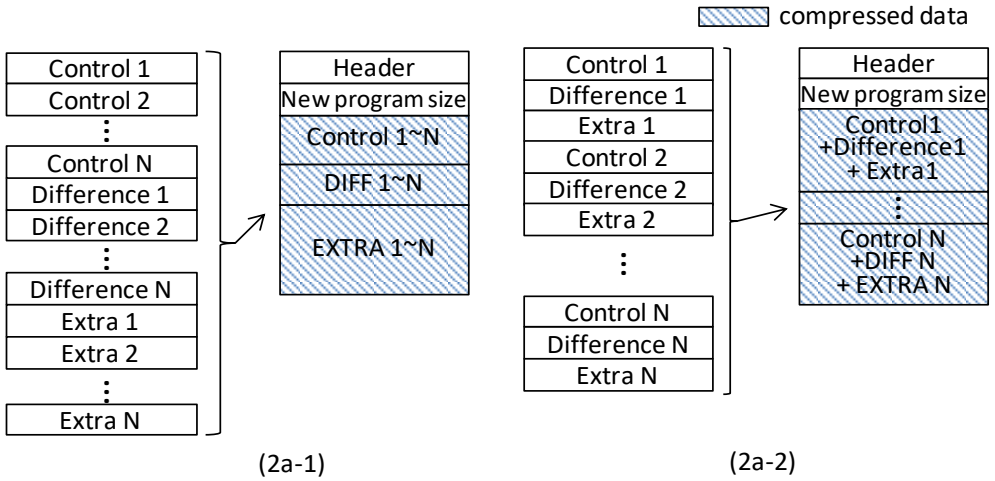


図 5-5 データ構成概略

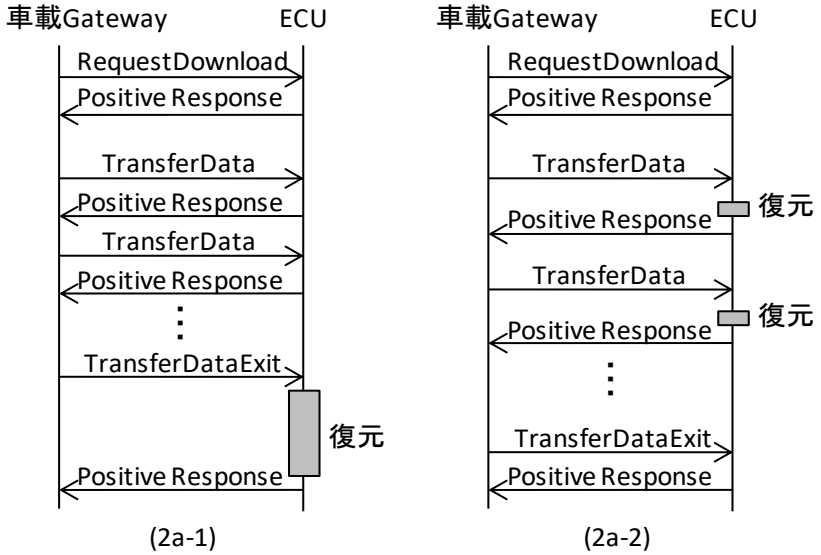


図 5-6 差分復元手順の比較

表 5-2 に本比較のまとめを示す．本研究では，メモリ使用効率を優先し，(2a-2)の方式を採用することとした．

表 5-2 データ構成の比較

方式	圧縮率	メモリ使用量
(2a-1)	○ 類似データが集合	× 一括復元
(2a-2)	△	○ 逐次復元可

(2a-2)の方式を用いて逐次復元を行う場合，ECU における復元処理は図 5-7 に示す手順で行う．

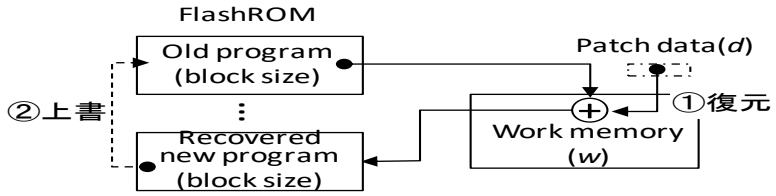


図 5-7 逐次復元処理

すなわち，固定長(例えば，256byte)の領域を差分格納領域 d として受信したデータを格納し，当該領域の残量がなくなった時点で復元を開始する．ワークメモリで復元された新プログラムの断片は，復元領域 n に格納し，1 ブロック分を復元したところで，対応する領域の旧プログラムを上書きする．ここでは，更新中の電源断等発生時にも旧プログラム・新プログラムのいずれかが存在する状態にして信頼性を向上させるため，復元領域 n は Flash ROM 上に確保する例を示した．

5.3.3. 圧縮アルゴリズム

bsdiff で利用されている bzip2 は，Burrows-Wheeler 変換[156]及び Move To Front 法によるブロックソートと，Huffman 符号[157]による符号化を組み合わせて deflate[158]等と比較して高い圧縮率を実現している．一方で，復元時間が遅い，復元時の消費メモリが多い，などの課題がある．そのため，前述のように圧縮を直列化したとしても，依然として 110Kbyte のワークメモリ w が必要となる．

そこで，本研究では，圧縮アルゴリズムとして圧縮効率が高く，復元が高速な LZMA[159]をベースとした 2 段階圧縮方式を適用する．すなわち，抽出した差分データに対し，LZ77[160]ベースの辞書式符号化に加え，Binary Range Coder[161]を利用して圧縮率の向上を図る(図 5-8)．

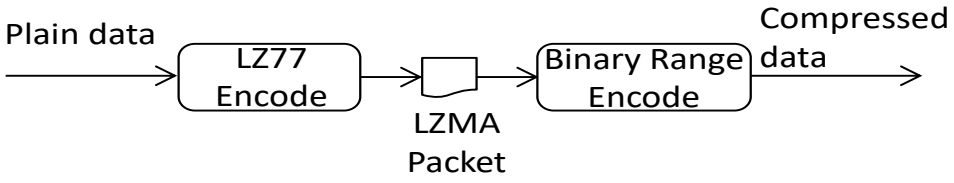


図 5-8 LZMA の圧縮処理

5.3.4. LZMA に基づく 2 段階圧縮方式

LZMA の辞書式符号化では，入力データは LZMA Packet に符号化される．LZMA Packet は，一致する文字列がある場合に，単純にオフセットと長さを表現するのではなく，一致が繰り返される場合にこれを短く表現できるように 7 種類の符号が定義されている．この LZMA Packet をさらに符号化する Binary Range Encode ではコンテキストに基づく確率情報を利用し，圧縮率を向上している．コンテキストとしては，符号化対象の Packet 種別に加え，直前の符号化の状態や符号対象の位置情報などを利用する．これにより圧縮率の向上が図れる一方で，確率情報管理のために必要なメモリが増大する．LZMA では，確率の管理に約 14Kbyte のメモリが必要となる．本提案では，利用する確率情報を限定し，必要なワークメモリ w を削減する．

5.3.5. 提案方式

以上より，車載 ECU への搭載性を考慮して復元時のメモリ使用量を削減した差分更新方式として，方式(1b)の広域ブロック差分方式，(2a-2)の直列化圧縮化方式，2 段階圧縮方式を適用した改良版 bsdiff を提案する．表 5-3 に bsdiff と提案方式の比較を示す．

表 5-3 bsdiff と提案方式の比較

項目	bsdiff	提案方式
差分生成単位	プログラム全体比較	広域ブロック差分(1b)
差分ファイル構造	並列圧縮	復元順序を考慮した データ直列化(2a-2)
圧縮アルゴリズム	bzip2	LZMA ベースの 2 段階圧縮方式

5.4. 評価

5.4.1. 評価環境

本研究では，実際の車両に搭載されている 4 種類の ECU 用プログラムと車載 ECU 向けのオープンソースソフトウェアを対象として評価を実施した．評価対象プログラムとサイズを表 5-4 に示す．

表 5-4 評価対象プログラムとサイズ

テスト プログラム	有効 ROM サイズ[byte]	備考
A	1,998,848	パワートレイン系
B	1,572,864	パワートレイン系
C	1,310,720	走行系
D	1,310,720	走行系
E	98,304	TOPPERS/ATK2[162] SC1 1.3.2→SC2 1.0.0

ここで，有効 ROM サイズとは，未使用のブロックを除いた，更新対象のプログラムが書き込まれているブロックの合計サイズを示す．

評価に用いた環境条件を図 5-9 及び表 5-5 に示す．

図 5-9 評価環境外観

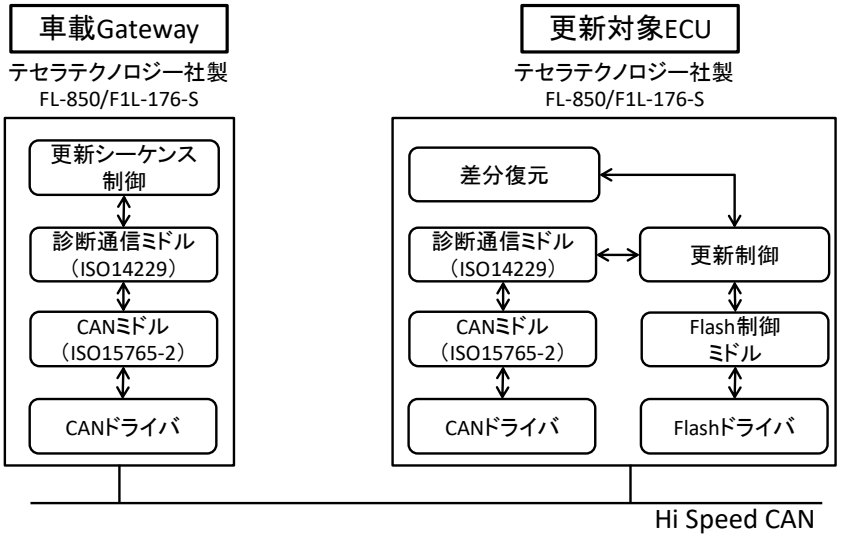


図 5-10 評価環境機能配置

表 5-5 評価環境

項目		値と説明	
CPU		80MHz	Renesas RH850/F1L
ROM		2MB	70 ブロックで構成 (8Kbyte×8, 32Kbyte×62)
RAM		160KB	
CAN	帯域	50kbps	Hi-SPEED CAN(500kbps)の帯域 10%利用
	プロトコル	ISO15765-2, ISO14229(UDS)	
	パラメータ	Separation time=0,Block size=1	

本研究では，テセラ・テクノロジー社製のマイコン評価ボード FL-850/F1L-176-S にルネサスエレクトロニクス社製の RH850/F1L を搭載し評価環境を構成した．本評価では，車載ゲートウェイ及び更新対象 ECU に同じ評価ボードを用いそれぞれに異なるソフトウェアを実装した．評価用車載ゲートウェイ及び更新対象 ECU の機能配置を図 5-10 に示す．車載ゲートウェイには UDS で規定された手順に従い更新シーケンスの制御を行う更新シーケンス制御部，UDS コマンドの構成や解析を行う診断通信ミドル，ISO15765-2 に対応した CAN 通信を行う CAN ミドル及び CAN ドライバを搭載する．また，更新対象 ECU には，車載ゲートウェイからの指示に従って差分復元部や診断通信ミドル，Flash 制御ミドルを制御して ECU ファームウェアの更新を制御する更新制御部と，差分復元処理を行う差分復元部，Flash ROM の消去や書込みを制御する Flash 制御ミドルと Flash ドライバ，UDS コマンドの構成や解析を診断通信ミドル，ISO15765-2 に対応した CAN 通信を行う CAN ミドル及び CAN ドライバを搭載する．Flash 書込み制御機能およびこれらを制御する更新制御機能を搭載する．CAN の通信帯域は 500Kbps であるが，通常ソフト更新のための診

断通信に割り当てられる帯域は少ないため，10%が利用可能であると仮定した．ここで，ISO15765-2 に規定された CAN の通信パラメータとして Separate time は 0，Block size は 1 を利用した．

評価項目としては，プログラム更新に必要なメモリ量，圧縮率，更新時間を評価した．表 5-6 に評価項目及び評価方法をまとめる．

本研究では， 従来方式の bsdiff/bspatch との比較を基本として提案方式の効果を検証する．また，広域ブロック差分の効果を検証するため，ブロック差分とも比較を行う．

表 5-6 評価項目と方法

評価項目	評価方法
メモリ量	ソースコード及びメモリマップを利用して積算
圧縮率	PC 上で圧縮アルゴリズムを実装. 評価対象プログラムの差分ファイルを生成し， サイズ評価
更新時間	RH850/F1L 環境での実測.

プログラム更新に必要なメモリ量は，ECU に差分復元ソフトを組み込んで動作させるために必要な ROM サイズ及び RAM サイズである．これらの必要量については，ソースコードおよびメモリマップからの積算により評価する．圧縮率は，差分生成アルゴリズムを実装して各評価対象プログラムの差分圧縮を行った際にどれだけデータを削減できたかを評価する．更新時間については，差分復元アルゴリズムを更新対象 ECU 上に実装して実測する．更新時間は，式(2)のように表現できる．

$$T_{update} = T_{GW} + T_{ECU} + T_{com} \qquad (2)$$

ここで，T_{GW}は車載ゲートウェイ内の処理時間，T_{ECU}は ECU 内処理時間，T_{com}は通信時間である．ECU 内の処理時間 T_{ECU}は，式(4)で算出する．

$$T_{ECU} = T_{decomp} + T_{Flash} \qquad (3)$$

ここで，T_{decomp} は ECU 上で差分を復元する時間であり，T_{Flash}は FlashROM の消去・書込みに要する時間である．本評価においては，車載ゲートウェイ側で T_{update}，T_{GW}を計測し，更新対象 ECU 側で T_{decomp}及び T_{Flash}を計測する．

CAN 通信時間 $Tcom$ はこれら計測結果を用いて式(3)で算出する．

$$T_{com} = T_{update} - (T_{GW} + T_{ECU}) \qquad (4)$$

前述の評価環境における通常更新時の更新時間計測結果を表 5-7 に示す.

表 5-7 通常更新時の更新時間[s]

Test program	T_{update}	T_{com}	T_{GW}	T_{ECU}		
					T_{decomp}	T_{flash}
A	1231.0	1215.7	1.1	14.2	0	14.2
B	970.1	957.9	0.8	11.2	0	11.2
C	808.2	798.1	0.7	9.4	0	9.4
D	808.0	798.0	0.7	9.4	0	9.4
E	47.0	46.3	0.1	0.6	0	0.6

実際の更新に当たっては，前述の時間に加えて，更新データをサーバからダウンロードする時間，受信したデータの暗号復号・検証等を行う時間が必要である．一方で，これらの処理はバックグラウンドで実施し，ユーザは意識しないように構成することが可能であるため，ここでは更新時間には含めない.

5.4.2. 評価

(1)使用メモリ量

表 5-8 に，提案方式のメモリ使用量の評価結果を示す.

表 5-8 提案方式のメモリ使用量[byte]

Item	bspatch([])	提案方式
プログラム(ROM)	16,350~*1	8,110
差分格納 d＋ワーク w(RAM)	862,208~*2	7,680
一時復元領域 (RAM または ROM)	32,768*3	32,768*3

*1:オリジナルの bspatch に加え，有向グラフ生成ロジックが必要
*2:有向グラフ生成・格納領域が必要
*3:インプレース更新における一時復元/退避領域(表 5-5 の評価環境における最大消去ブロックサイズ)

本評価においては，bspatch では，d=500Kbyte とし，bzip2 のブロックサイズを 4Kbyte とした．また，提案方式の辞書式符号化用 Window Size は 4Kbyte とした.

提案方式について，図 5-10 の更新制御部及び差分復元部のプログラムサイズは約 8KB, RAM 上の差分格納領域及びワークメモリは約 7.5KB であるため，例えば，評価環境とした RH850/F1L 等の車載 ECU に搭載可能なレベルのメモリ使用量で差分を復元できることを確認できた．なお，これらのメモリ使用量は，入力データの内容に依存しない．なお，広域ブロック差分・ブロック差分では，参照先が変わるだけのため復元時のメモリ使用量は変

わらない．差分データと旧プログラムから再構成した新プログラムを一時的に格納する一時格納領域は，RAM 上，ROM 上いずれに用意してもよい．更新のロバスト性を向上するためには ROM 上に確保することが望ましいが，Flash ROM 寿命への影響や更新速度への影響を考慮する必要がある．

(2)圧縮率

前述の評価対象プログラムについて，バージョン間の差分を生成して元プログラムとの比較を行い，圧縮率を評価する．圧縮率 $r_{compression}$ は式(5)で計算する．

$$r_{compression} = S_d/S_R \qquad (5)$$

ここで，S_d は生成されたパッチファイルのサイズ，S_R は有効 ROM サイズである．表 5-9 にオリジナルの bsdiff，ブロック差分方式及び広域ブロック差分方式にて生成したパッチファイルのサイズを示す．また，各方式の圧縮率の比較を図 5-11 に示す．

表 5-9 提案方式のパッチファイルサイズ S_d[byte]

テスト プログラム	bsdiff	ブロック 差分	広域ブロック 差分
A	12,929	24,617	23,090
B	86,640	130,054	100,549
C	146,740	351,473	157,279
D	37,509	73,951	45,178
E	8,132	8,167	7,671

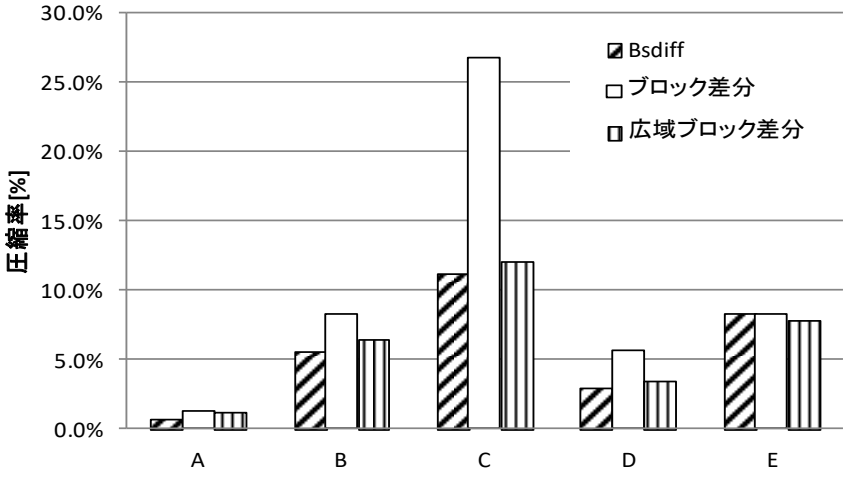


図 5-11 提案方式の圧縮率

提案方式ではメモリ使用量を削減できた一方で，ブロック単位に分割して差分を生成し

たことによって，圧縮率が低下している．特に，ブロック差分方式では，C のケースで約15%圧縮率が低下している．これは，C のケースでは新規に追加されたコード部分が大きかったためである．しかし，広域ブロック差分を適用した場合は，評価したすべてのケースにおいて圧縮率の低下が 1%以内であった．これにより，提案方式では ECU に適用可能なサイズにメモリ使用量を削減しつつ，圧縮性能を維持できていることを確認できた．

(3)更新時間の評価

更新時間の評価は，圧縮率の評価の結果もっとも効果の高かった広域ブロック差分で生成したデータについて，評価を実施した．本評価では，一時復元領域は RAM 上に確保した．また，従来手法による復元処理は，表 5-8 に示したとおり，RAM が 160KB の車載 ECU 向けマイコンを用いた本評価環境ではメモリ不足のため搭載できない．このため，更新時間時間の評価は，提案手法についてのみ実施した．

表 5-10 に，本提案による差分更新時の更新時間の評価結果を示す．

表 5-10 差分更新適用時の更新時間[s]

Test program	T_{update}	T_{com}	T_{GW}	T_{ECU}		
					T_{decomp}	T_{Flash}
A	**28.7**	**14.8**	0.1	13.8	1.7	12.1
B	**73.7**	**61.8**	0.1	11.8	1.8	10.0
C	**105.8**	**96.2**	0.2	9.5	1.7	7.8
D	**37.1**	**28.0**	0.1	9.0	1.2	7.8
E	**5.5**	**4.7**	0.04	0.8	0.1	0.7

差分更新の適用により CAN を転送するデータ量が削減されるため，通信時間 Tcom を 1/10 以下に短縮できる．一方で，差分更新時に必要となる差分復元時間 T_{decomp} についても，最大で 1.8 秒であり，すべての評価プログラムにおいて更新時間全体の 1/10 以下である．これにより，本提案方式による差分復元時間は，更新に必要な時間に対して十分に小さいことが確認できた．表 5-11 に通常更新及び本提案による差分更新における更新時間の比較を示す．

表 5-11 更新時間の比較[s]

テスト プログラム	通常	広域ブロック差分
A	1231.0	28.7(2.3)
B	970.1	73.7(7.6)
C	808.2	37.1(4.6)
D	808.0	105.8(13.1)
E	47.0	5.5(11.7)

通常更新時と差分更新時の更新時間の比較により，差分復元時間を含めても，差分更新を適用することによって更新時間を短縮できることが確認できた．

5.5. 結語

本章では，bsdiff をベースに ECU に適用可能な差分更新方式を提案した．提案方式では，ブロック単位の差分生成とデータ構造及び圧縮アルゴリズムの変更により，圧縮率を維持しつつ，ECU に搭載可能なレベルへの省メモリ化を実現した．具体的には，評価したテストケースについては圧縮率低下 1%以内で，入力データの内容によらず約 7.5KB の RAM での差分復元を可能とした．また，提案方式を自動車向けマイコン上に実装し ECU に搭載可能なレベルに省メモリ化できたことを確認した．さらに，実機評価により，本方式を適用することで，ECU ソフトウェアの更新時間を短縮できることを確認した．

以上により，リソース制約のあるノードに対して差分更新を適用するための手法について示した．具体的には，差分生成時に復元時のメモリ使用量を考慮してデータ構造，差分データの生成単位，圧縮アルゴリズムを選定することでプログラムサイズに対してノードの RAM サイズが小さい場合にも差分更新を適用可能であることを示した．

2.4.1 に述べた通り，車載 ECU 用のマイコンは高機能化し，利用可能なメモリリソースも増加している．また，ネットワークの広帯域化のため，Ethernet の導入が始まっている．このような環境においては，提案方式を用いなくても短時間での更新が可能となる．他にも，プログラムを格納する Flash メモリを 2 面持ち，1 面動作中に動作していないもう一面に新しいプログラムの書き込みが可能なアーキテクチャのマイコンも提案されており，この場合は，差分更新により更新時間を短縮しなくてもシステムの可用性に与える影響は小さくなると考えられる．一方で，高機能なマイコンや Ethernet の導入にはコストがかかることから，その範囲は必要最小限にとどめられ，多くの ECU は，従来通りリソースを抑えた構成となると見込まれる．例えば，図 2-14 に示したドメインコントローラを高機能マイコンや SoC で構成し，その間を高速な Ethernet で接続，ドメイン内の ECU やネットワークは低コストな部品を採用するような構成となっていく．このような観点から，省リソースで更新時間の短縮ができる本提案の技術は次世代のアーキテクチャにも有効である．

さらに，産業用途で用いられるセンサネットワークなどのシステムにおいても，狭帯域な環境で省リソースに更新を実現することは重要である．バッテリで駆動し，間欠的に無線通信を行うような端末では，消費電力を抑えるために動作周波数やメモリリソース，ネットワーク帯域に制限がある．さらに，図 1-4 に示したように，IoT デバイスの消費電力の増加が将来の課題となり得ることから，将来にわたっても，低消費電力で動作できるようにリソースやネットワーク帯域を抑えた組込みシステムは，拡大していく IoT システムの中で大き

な位置づけを占めると考えられる．このような観点から，本提案の技術は，自動車システムだけでなく，広くネットワーク接続された組込みシステムにおいても有効であると考える．

第6章 結論

6.1. まとめ

本論文では，ネットワーク接続された組込みシステムの拡張に関して，システムのキーコンポーネントとなるゲートウェイ機能のソフトウェア構成方法に関する提案と，システムを構成するノードとしての組込みシステムのソフトウェア更新方法に関する提案を行った．具体的には，ネットワーク化システムの拡張に資する技術として，制御機能の集中配置型システムにおけるゲートウェイ機能の標準プロトコル向けミドルウェアの設計手法，ネットワーク上に多様なノードが接続されるシステムにおけるノードのソフトウェア更新用制御機能の設計手法，そしてリソース制約のある環境での場合のノード自身の更新手法を提案した．

1 章では，爆発的に増大している IoT システムの動向とこれを構成する組込みシステムの特性について述べるとともに，IoT システムの特徴がシステム拡張にあることを示した．そのうえで，今後の IoT デバイスの成長率をもとに，今後増加すると見込まれる，ノードのインターネット接続に制限があるネットワーク化された組込みシステムのシステム拡張が重要であることを示し，本研究の目的を前記システムの拡張方式の提案によるネットワーク化された組込みシステムの価値向上の容易化とした．さらに，目的を実現するために，IoT デバイス数の多いホームネットワークシステムと，今後の高い成長が見込まれる自動車システムにおいて，システム拡張を容易にする構成方法に関する課題の解決およびシステム拡張を実行する過程に関する課題の解決により，システム拡張の課題を解決するアプローチをとることを示した．

2 章では，ネットワーク接続された組込みシステムの拡張に関する関連研究を体系的に整理した．最初に，システム拡張に必要な技術を，システム拡張を可能にする，または容易にする構成に関する技術とシステム拡張を実行する過程に関する技術に分類し，それぞれの先行技術を示した．システム拡張を可能または容易にする構成に関する技術においては，ハードウェアの再構成技術及び再構成を考慮したソフトウェアの構成に関する技術を示し，ソフトウェアの構成に関する技術としては，再構成の前提となる OS・バーチャルマシンレイヤの技術，再構成を容易にするミドルウェアやフレームワーク技術を整理した．システム拡張を実行する過程に関する技術においては，ノードの再構成のための更新データの生成と配送に関する技術と，追加または更新されたノードのシステムへの統合に関する技術について述べた．前者については配信データの作成技術，配信管理及びノードへの配信技術，ノードにおける更新適用技術について述べ，後者については，サービスの検出技術について

述べた．さらに，本研究の対象とするホームネットワークシステムと自動車システムについてその基本構成と関連技術を示し，システム拡張における課題をまとめた．

3 章では，ホームネットワークシステムを対象に，コントローラとなるサービスゲートウェイに搭載するミドルウェアの設計方法を論じた．国内で公知な標準インタフェースとして推奨された ECHONET Lite に対応したサービスゲートウェイ向けのミドルウェアを OSGi のバンドルとして開発した．設計に当たり，マルチサービスを実現するサービスゲートウェイ上にミドルウェアを実装するにあたっての課題及び ECHONET Lite 規格の実装上の課題を検討した．検討した課題に基づいて，OSGi フレームワークの機能を最大限活用して，アプリケーション開発工数の低減，省リソースとアプリケーション停止時間最小化の両立，任意の ECHONET Lite 機器特定の容易化，下位通信層への柔軟な対応を実現する ECHONET Lite バンドルの構成を提案した．さらに，提案する構成のバンドルを実装して評価システムを構築し，前記要件の観点から提案する ECHONET Lite バンドルの有効性を確認した．

4 章では，自動車システムを対象に，OTA による ECU のソフトウェア更新のための更新制御機能の設計方法を論じた．OTA による遠隔更新では，更新処理の自動化が必要でありそのための更新制御技術が重要となる．本研究では，自動車システム向けの OTA 更新制御についての要件を整理し，整理した要件を満たす機能配置を提案した．また，自動車システム特有の多様性に対応した OTA システムの構成と更新制御方式を提案した．提案方式を車載ゲートウェイ用マイコン上に実装し，提案方式が有効であることを確認した．

5 章では，自動車システムを構成する ECU を対象に，そのリソースを考慮してソフトウェア更新時間を短縮する差分更新方式について述べた．OTA による遠隔更新では，更新時間の短縮が重要となりそのための要素技術として差分更新が必要になる．本研究では，ECU に適用可能な差分更新方式を提案した．また，提案方式を，自動車向けマイコン RH850 上に実装し，当該マイコン上に搭載可能なレベルに消費メモリを削減し，更新時間を短縮できることを確認した．

以上のように，本論文では，ネットワーク接続された組込みシステムの代表としてのホームネットワークシステムおよび自動車システムにおいて，多様なノードから構成されるシステムのゲートウェイのソフトウェア構成方法と省リソースなノードのソフトウェア更新方法を示し，これがノードの追加及びノードの再構成によるシステム拡張に有効であることを確認した．2 章で述べた自動車システムと同様，他の分野で用いられる組込みシステムのリソースも全体としては増加傾向にある一方で，デバイス数の増加による IoT デバイスの消費電力の増大が課題となりつつある．また，システムの普及にはシステムコストの低減は重要な要素である．このことから，省リソースで低消費電力・低コストに，多様なノード

から構成されるシステムの拡張を可能とする本研究の成果の適用により，ネットワーク化された組込みシステムの価値向上に有用であると考える．

6.2. 今後の展望

本論文では，ソフトウェア更新によるシステム拡張について論じ，その際の課題として，組込みシステムの一般的な特性であるリソース制約などを考慮した設計手法を提案した．ネットワーク接続された組込みシステムへのインターネットへの接続は，今後ますますの増加が予想されており，ソフトウェア更新によるシステム拡張はそれに伴って加速するものと考えられる．

例えば，自動車システムにおいては，遠隔でのソフトウェアの更新は今後実用化されて，不具合の修正やセキュリティパッチの適用など，安全性の向上に必須となるとともに，機能の追加など付加価値の向上に貢献する技術となる．さらなる進展としては，付加価値の向上として，スマートハウスなど車両外部のシステムとの連携機能や，さらなる安全性の向上のため，システムの失陥時に動的なシステム再構築などにも利用されていくものと考えられる．このためには，安全状態を保ちつつ，自動車のソフトウェア更新を動的に行うような検討を行う必要がある．また，自動運転システムの制御に，Deep Learning などの機械学習技術を適用する際には，収集したデータから学習により改善したアルゴリズムを自動車にフィードバックするなど，システム拡張技術を AI との連携させることで，進化する車両の実現が期待できる．

さらに，第 1 章で述べた通り，このような組込みシステムへのフィードバックは，IoT 化されるあらゆる分野で重要な役割を担うようになる．例えば，多様なセンサや機械から成る工場システムのシステム拡張による生産性の向上やリモートメンテナンスと組み合わせた制御方法の改善による製品寿命の延長など，様々な活用シーンへの適用が可能である．

このような環境を実現するためには，組込みシステムとクラウド側の更なる結合が重要となる．例えば，本論文で論じた差分更新を適用するためには，組込みシステム側のソフトウェア構成を正確に把握するための構成管理技術が必須である．また，機械学習などシステム拡張が個別システムごとに行われるようになる場合も，それぞれのシステムの環境などを把握するためにシステム構成やデータなどをクラウド側で収集・複製できることが重要となる．さらに，組込みシステムとクラウド側の密結合の形態として，スマートスピーカのようにアプリケーションの主要機能がクラウド側に配置されるような構成も進展していくと考えられる．2.4.1 節で述べたように，自動車システムにおいてもクラウド側への機能の再配置が進展する可能性がある．

今後は，このような検討を進めることで，システム拡張・再構築により，さらなる付加価

値・安全性の向上を進めていく.

謝辞

本論文は，著者が京都大学大学院情報学研究科知能情報学専攻博士後期課程および株式会社日立製作所　横浜研究所に在籍中の研究成果をまとめたものです．指導教員である岡部寿男教授には，本論文の構成にあたって多大なるご指導をいただき，深く感謝申し上げます．また，副査である通信情報システム専攻　高木直史教授，五十嵐淳教授には，審査を通じて本論文へのご指導をいただき，深く感謝申し上げます．知能情報学専攻　宮崎修一准教授，小谷大祐助教をはじめとする岡部研の皆様には本論文に関する研究について、研究会などを通じてご助言をいただき感謝申し上げます．

株式会社日立製作所および日立オートモティブシステムズ株式会社の職場の皆様には，本論文執筆のきっかけを与えてくださるとともに，研究の指針や技術課題に関して適切なご助言・ご指導をいただくとともに、論文の執筆や学位取得の活動をさまざまな面でサポートいただき深く感謝申し上げます．

最後に，本論文の執筆にあたり，常に生活を支えてくれた妻と子供たちに感謝いたします．

参考文献

[1] 総務省. 情報通信白書平成 29 年版

[2] 日本学術会議　日本の展望―学術からの提言 2010　情報学分野の展望

[3] Heath, Steve (2003). Embedded systems design. EDN series for design engineers (2 ed.).

[4] 中本幸一, 高田広章, & 田丸喜一郎. 組込みシステム技術の現状と動向. 情報処理学会論文誌. 1997, vol. 38, no. 10, p. 1-8.

[5] 南角茂樹, 高田広章, 岸田昌巳, & 宿口雅弘. “組込みシステム概論”. リアルタイム OS と組み込み技術の基礎：実践μITRON プログラミング. p. 7-16.

[6] Friedli, M., Kaufmann, L., Paganini, F., & Kyburz, R. (2016). Energy efficiency of the internet of things. Technology and Energy Efficiency Report, IEA 4E EDNA.

[7] IPA. 組込みソフトウェア開発データ白書 2017.

[8] 近藤満. (2007). 組込みソフトウェア開発の課題と対応. 赤門マネジメント・レビュー, 6(10), 493-502.

[9] 高田広章. 組込みシステム開発技術の現状と展望. 情報処理学会論文誌. 2001, vol. 42, no. 4, p. 930-938

[10] Gupta, R. A., & Chow, M. Y. (2010). Networked control system: Overview and research trends. IEEE transactions on industrial electronics, 57(7), 2527-2535.

[11] Glaessgen, E., & Stargel, D. (2012, April). The digital twin paradigm for future NASA and US Air Force vehicles. In 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA (p. 1818).

[12] Grieves, M. (2014). Digital twin: manufacturing excellence through virtual factory replication. White paper.

[13] Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H., & Sui, F. (2018). Digital twin-driven product design, manufacturing and service with big data. The International Journal of Advanced Manufacturing Technology, 94(9-12), 3563-3576.

[14] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys & Tutorials, 17(4), 2347-2376.

[15] IPA. 組込みソフトウェアを用いた 機器におけるセキュリティ （ 改訂版 ）. 2010.

[16] 清原良三. (2008). 携帯電話のソフトウェアアドインに関する研究. https://ir.library.osaka-u.ac.jp/repo/ouka/all/2273/22506_%E8%AB%96%E6%96%87.pdf

[17] Boie, A. (2015, March). Android* Software Updates. In Embedded Linux Conference.

[18] https://www.nintendo.co.uk/Support/Wii/Usage/Wii-Menus/Wii-System-Update/Wii-System-Update-242935.html

[19] https://www.jp.playstation.com/ps4/system-update/ (2019-1-20 参照)

[20] https://www.apab.or.jp/receiver/es.html (2019-01-20 参照)

[21] 末吉敏則, & 飯田全広. (1999). やわらかいハードウェア: リコンフィギャラブル・コンピューティング. 情報処理, 40(8).

[22] Bobda, C. (2007). Introduction to reconfigurable computing: architectures, algorithms, and applications. Springer Science & Business Media.

[23] Balani, R., Han, C. C., Rengaswamy, R. K., Tsigkogiannis, I., & Srivastava, M. (2006, October). Multi-level software reconfiguration for sensor networks. In Proceedings of the 6th ACM & IEEE International conference on Embedded software (pp. 112-121). ACM.

[24] OSGi Alliance「OSGi Service Platform Core Specification」 http://www.osgi.org/(2012-12-13 参照)

[25] Gong, L. (2001). A software architecture for open service gateways. IEEE Internet computing, 5(1), 64-70.

[26] Li, Y., Wang, F. Y., He, F., & Li, Z. (2005, June). OSGi-based service gateway architecture for intelligent automobiles. In Intelligent Vehicles Symposium, 2005. Proceedings. IEEE (pp. 861-865). IEEE.

[27] 角田潤, 内藤壮司, & 松倉隆一. (2012). ネットワークレディ機器の活用を加速するサービスプラットフォーム技術. Fujitsu, 63(6), 675-680.

[28] ECHONET コンソーシアム「ECHONET 規格書 Version 3.21」 http://www.echonet.gr.jp (2019-2-21 参照)

[29] Moon, K. D., Lee, Y. H., Son, Y. S., & Kim, C. K. (2003). Universal home network middleware guaranteeing seamless interoperability among the heterogeneous home network middleware. IEEE Transactions on Consumer Electronics, 49(3), 546-553.

[30] ECHONET Lite 規格書 https://echonet.jp/spec_v113_lite/

[31] 宮丸卓也, 峰野博史, 寺島美昭, 徳永雄一, & 水野忠則. (2008). センサネットワークにおける無線通信を利用した効率的ソフトウェア配布方式. 情報処理学会研究報告マルチメディア通信と分散処理 (DPS), 2008(21 (2008-DPS-134)), 183-188.

[32] Nakanishi, T., Shih, H. H., Hisazumi, K., & Fukuda, A. (2013, May). A software update scheme by airwaves for automotive equipment. In Informatics, Electronics

& Vision (ICIEV), 2013 International Conference on (pp. 1-6). IEEE.

[33] de Boer, Gerrit, Peter Engel, and Werner Praefcke. "Generic remote software update for vehicle ecus using a telematics device as a gateway." Advanced Microsystems for Automotive Applications 2005. Springer Berlin Heidelberg, 2005. 371-380.

[34] Baic, D., Langjahr, P., Haas, W., & Fessard, A. (2018). Safe computing with central ECUs. In 18. Internationales Stuttgarter Symposium (pp. 155-163). Springer Vieweg, Wiesbaden.

[35] UNECE WP.29 GRVA.(2018).Draft Recommendation on Software Updates of the Task Force on Cyber Security and Over-the-air issues of UNECE WP.29 GRVA

[36] 国土交通省　自動運転等先進技術に係る制度整備小委員会　Web サイト http://www.mlit.go.jp/policy/shingikai/s304_jidouunten01.html (2018/12/23 参照)

[37] http://grouper.ieee.org/groups/802/3/ (2019-1-20 参照)

[38] WiFi Alliance https://wi-fi.org/ (2019-1-20 参照)

[39] https://openconnectivity.org/developer/specifications/upnp-resources/upnp(2018-12-23 参照)

[40] Digital Living Network Alliance https://www.dlna.org/(2019-1-20 参照)

[41] https://www.profibus.com (2019-1-20 参照)

[42] http://www.modbus.org (2019-1-20 参照)

[43] https://www.odva.org/Technology-Standards/EtherNet-IP/Overview (2019-1-20 参照)

[44] Bosch, R. (1991). CAN specification version 2.0. Rober Bousch GmbH, Postfach, 300240.

[45] ISO 17987 Part 1-7

[46] https://www.mostcooperation.com/ (2019-1-20 参照)

[47] ISO 17458-1:2013 Road vehicles -- FlexRay communications system -- Part 1: General information and use case definition

[48] http://www.opensig.org/(2019-1-20 参照)

[49] 木谷光博, 片岡幹雄, & 寺岡秀敏. (2016). 自動運転向け車内ネットワークシステムにおけるデータ伝送方式の開発. 情報処理学会論文誌コンシューマ・デバイス & システム (CDS), 6(2), 43-51.

[50] ISO 15765-2:2011 Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) — Part 2: Transport protocol and network layer services

[51] ISO 14229-1:2013 Road vehicles -- Unified diagnostic services (UDS) -- Part 1: Specification and requirements

[52] 佐藤浩一, 豊山祐一, 田中誠, 越塚登, & 坂村健. (2004). 組込みシステムのプラットフォームの標準化によるソフトウェア資産の再利用性向上の評価. 第 66 回全国大会講演論文集, 2004(1), 19-20.

[53] https://www.autosar.org/(2019-1-20 参照)

[54] 吉村健太郎. (2007). 製品間を横断したソフトウェア共通化技術-ソフトウェアプロダクトラインの最新動向. 情報処理, 48(2), 171-176.

[55] 佐々木隆益, 吉岡信和, 田原康之, & 大須賀昭彦. (2016). 組込み向け進化型ソフトウェアの効率的な拡張性強化手法. 情報処理学会論文誌, 57(2), 730-744.

[56] Compton, K., & Hauck, S. (2002). Reconfigurable computing: a survey of systems and software. ACM Computing Surveys (csuR), 34(2), 171-210.

[57] Shoa, A., & Shirani, S. (2005). Run-time reconfigurable systems for digital signal processing applications: A survey. Journal of VLSI signal processing systems for signal, image and video technology, 39(3), 213-235.

[58] Zynq UltraScale+ MPSoC https://japan.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[59] 日経 BP 社.やわらかくなる LSI--広がり始めた動的再構成 .日経エレクトロニクス (2011), 59-66, 2011-08-22

[60] Han, C. C., Rengaswamy, R. K., Shea, R., Kohler, E., & Srivastava, M. (2005, June). SOS: A dynamic operating system for sensor networks. In Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys) (pp. 1-2).

[61] Dunkels, A., Gronvall, B., & Voigt, T. (2004, November). Contiki-a lightweight and flexible operating system for tiny networked sensors. In Local Computer Networks, 2004. 29th Annual IEEE International Conference on (pp. 455-462). IEEE.

[62] https://www.toppers.jp/(2019-1-20 参照)

[63] https://java.com/ja(2019-1-20 参照)

[64] https://developer.android.com/guide/platform/

[65] Simon, D., Cifuentes, C., Cleal, D., Daniels, J., & White, D. (2006, June). Java™ on the bare metal of wireless sensor devices: the squawk Java virtual machine. In Proceedings of the 2nd international conference on Virtual execution environments (pp. 78-88). ACM.

[66] Levis, P., & Culler, D. (2002, October). Maté: A tiny virtual machine for sensor networks. In ACM Sigplan Notices (Vol. 37, No. 10, pp. 85-95). ACM.

[67] Koshy, J., & Pandey, R. (2005, November). VMSTAR: synthesizing scalable runtime environments for sensor networks. In Proceedings of the 3rd international

conference on Embedded networked sensor systems (pp. 243-254). ACM.

[68] https://www.lua.org/ (2019-02-21 参照)

[69] Tanaka, K., Nagumanthri, A. D., & Matsumoto, Y. (2015, June). mruby--Rapid Software Development for Embedded Systems. In Computational Science and Its Applications (ICCSA), 2015 15th International Conference on (pp. 27-32). IEEE.

[70] 倉光君郎. (2010). 拡張性のある組み込みアプリケーションを実現するスクリプティング言語の開発. 情報処理学会論文誌, 51(12), 2185-2194.

[71] Dunkels, A. (2006). A low-overhead script language for tiny networked embedded systems. SICS Research Report.

[72] Sun, Y., Huang, W. L., Tang, S. M., Qiao, X., & Wang, F. Y. (2007, December). Design of an OSEK/VDX and OSGi-based embedded software platform for vehicular applications. In Vehicular Electronics and Safety, 2007. ICVES. IEEE International Conference on (pp. 1-6). IEEE.

[73] Park, P., Yim, H., Moon, H., & Jung, J. (2009, July). An OSGi based in-vehicle gateway platform architecture for improved sensor extensibility and interoperability. In Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International (Vol. 2, pp. 140-147). IEEE.

[74] 宮田克也, 寺岡秀敏, 関原拓也, & 芳野明彦. (2013). ホームゲートウェイ向け機器管理制御フレームワークの開発. 情報処理学会論文誌コンシューマ・デバイス & システム (CDS), 3(3), 39-48.

[75] Dixon, C., Mahajan, R., Agarwal, S., Brush, A. J., Lee, B., Saroiu, S., & Bahl, P. (2012, April). An operating system for the home. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (pp. 25-25). USENIX Association.

[76] Dafang, W., Shiqiang, L., Bo, H., Guifan, Z., & Jiuyang, Z. (2010, May). Communication mechanisms on the virtual functional bus of AUTOSAR. In Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on (Vol. 1, pp. 982-985). IEEE.

[77] Belaggoun, A., & Issarny, V. (2016, September). Towards adaptive autosar: a system-level approach. In FISITA 2016 World Automotive Congress.

[78] 伊賀徳寿, 中本幸一, 奥山嘉昭, 佐藤直樹, & 檜原弘樹. (2003). 組込みシステム向け CORBA の開発と評価. 情報処理学会論文誌コンピューティングシステム (ACS), 44(SIG10 (ACS2)), 164-176.

[79] 久保田稔. (2006). 動的適応性をもつモジュラー型基盤ソフトウェアの提案. 情報処理学会研究報告システム LSI 設計技術 (SLDM), 2006(28 (2006-SLDM-124)), 97-102.

[80] Ewing, M., & Troan, E. (1996, February). The rpm packaging system. In Proceedings of the First Conference on Freely Redistributable Software, Cambridge, MA, USA.

[81] Open Mobile Alliance http://www.openmobilealliance.org/wp/Overviews/dm_overview.html(2016-04-16 参照)

[82] Tridgell, A., & Mackerras, P. (1996). The rsync algorithm.

[83] TR-069 CPE WAN Management Protocol,Issue: 1 Amendment 6,Approval Date: March 2018,CWMP Version: 1.4

[84] Lightweight M2M – Software management Object (LwM2M Object – SwMgmt) Approved Version 1.0 – 01 Mar 2018

[85] http://www.pocketsoft.com/(2019-02-21 参照)

[86] http://www.redbend.com/en(2016-04-16 参照)

[87] http://www.daemonology.net/bsdiff/(2019-2-21 参照)

[88] http://xdelta.org/ (2019-2-21 参照)

[89] http://cis.poly.edu/zdelta/ (2015-03-16 参照)

[90] http://code.google.com/p/open-vcdiff/ (2019-2-21 参照)

[91] Wang, Q., Zhu, Y., & Cheng, L. (2006). Reprogramming wireless sensor networks: challenges and approaches. IEEE network, 20(3), 48-55.

[92] Tsiftes, N., Dunkels, A., & Voigt, T. (2008, June). Efficient sensor network reprogramming through compression of executable modules. In Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON'08. 5th Annual IEEE Communications Society Conference on (pp. 359-367). IEEE.

[93] Stolikj, M., Cuijpers, P. J., & Lukkien, J. J. (2012). Energy-aware reprogramming of sensor networks using incremental update and compression. Procedia Computer Science, 10, 179-187.

[94] Stolikj, M., Cuijpers, P. J., & Lukkien, J. J. (2013, March). Efficient reprogramming of wireless sensor networks using incremental updates. In Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on (pp. 584-589). IEEE.

[95] Lanigan, P. E., Gandhi, R., & Narasimhan, P. (2006). Sluice: Secure dissemination of code updates in sensor networks. In Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on (pp. 53-53). IEEE.

[96] Nilsson, D. K., & Larson, U. E. (2008, May). Secure firmware updates over the air in intelligent vehicles. In Communications Workshops, 2008. ICC Workshops' 08.

IEEE International Conference on (pp. 380-384). IEEE.

[97] Karthik, T., Brown, A., Awwad, S., McCoy, D., Bielawski, R., Mott, C., & Cappos, J. (2016). Uptane: Securing software updates for automobiles. In International Conference on Embedded Security in Car (pp. 1-11).

[98] 永田和生, 原田英雄, 牛嶋和行, 久我守弘, & 末吉敏則. (2007). FPGA 遠隔再構成システムの設計と実装. 電子情報通信学会論文誌 D, 90(6), 1357-1366.

[99] Simple Service Discovery Protocol/1.0 https://tools.ietf.org/html/draft-cai-ssdp-v1-03(2019-1-20 参照)

[100] Bonjour https://developer.apple.com/bonjour/(2019-1-20 参照)

[101] Edwards, W. K. (2006). Discovery systems in ubiquitous computing. IEEE Pervasive Computing, 5(2), 70-77.

[102] Helal, S., Desai, N., Verma, V., & Lee, C. (2003, March). Konark-a service discovery and delivery protocol for ad-hoc networks. In Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE (Vol. 3, pp. 2107-2113). IEEE.

[103] Klauck, R., & Kirsche, M. (2012, July). Bonjour contiki: A case study of a DNS-based discovery service for the internet of things. In International Conference on Ad-Hoc Networks and Wireless (pp. 316-329). Springer, Berlin, Heidelberg.

[104] Scalable service-Oriented MiddlewarE over IP (SOME/IP) http://some-ip.com/

[105] 丹康雄. (2010). ホームネットワーク（OSGi, ECHONET）モデルに基づく家庭内エネルギーマネジメント（制御・通信プロトコル,< 特集> エネルギーの情報化~ IT による電力マネジメント~). 情報処理, 51(8), 959-965.

[106] 「平成 23 年度 エネルギー管理システム導入促進事業（HEMS 導入事業）－対象機器の公募－ 公募要領 」

[107] 一般社団法人　エコーネットコンソーシアム　https://echonet.jp/

[108] IEC 62394 Ed3.0

[109] ISO/IEC 14543-4-3

[110] Echonet コンソーシアム 20 周年記念誌 https://echonet.jp/wp/wp-content/uploads/pdf/General/Download/20th_book_ver.0.7.pdf

[111] 大和ハウス工業株式会社「平成 21 年度スマートハウス実証プロジェクト報告書 第 2 章 テーマ 2-1：マッシュアップを促進するホームサーバ向け統合 API の開発実証およびテーマ 3-1：マルチベンダによる家電・設備機器統合コントロールシステムの開発」, 2010 年 3 月
http://www.jipdec.or.jp/dupc/forum/eships/results/doc/h21project_report1-2.pdf

[112] Fukushima, K.; Tanaka, Y.; Kato, H.; Ishikawa, N.; , "Home Network System for Gas Appliances Using PUCC Technologies," Consumer Communications and

Networking Conference (CCNC), 2010 7th IEEE , vol., no., pp.1-5, 9-12 Jan. 2010

[113] 日新システムズ EW-ENET Lite http://www.co-nss.co.jp/p-org/enetlite.html (2012-12-13 参照)

[114] SonyCSL「OpenECHO」 https://github.com/SonyCSL/OpenECHO(2012-12-13 参照)

[115] 富士キメラ総研ニュースリリース http://www.group.fuji-keizai.co.jp/press/pdf/170517_17042.pdf (2019-02-21 参照)

[116] Miller, C., & Valasek, C. (2015). Remote exploitation of an unaltered passenger vehicle. Black Hat USA, 2015.

[117] Foster, I. D., Prudhomme, A., Koscher, K., & Savage, S. (2015, August). Fast and Vulnerable: A Story of Telematic Failures. In WOOT.

[118] M.Kirsten, and T.Königseder. Automotive Ethernet. Cambridge University Press, 2014

[119] Challenges in a Future IP/Ethernet-based In-Car Network for Real-Time Applications

[120] https://www.renesas.com/jp/ja/img/misc/catalogs/pdf/r30ca0007ej0450-automotive.pdf(2019-02-21 参照)

[121] Liu, S., Tang, J., Zhang, Z., & Gaudiot, J. L. (2017). Computer Architectures for Autonomous Driving. Computer, 50(8), 18-25.

[122] Otterness, N., Yang, M., Rust, S., Park, E., Anderson, J. H., Smith, F. D., ... & Wang, S. (2017, April). An evaluation of the NVIDIA TX1 for supporting real-time computer-vision workloads. In Real-Time and Embedded Technology and Applications Symposium (RTAS), 2017 IEEE (pp. 353-364). IEEE.

[123] Schmidgall, R. (2012). Automotive embedded systems software reprogramming (Doctoral dissertation, Brunel University School of Engineering and Design PhD Theses).

[124] https://www.teslamotors.com/sites/default/files/pdfs/release_notes/tesla_model_s_software_7_1.pdf(2019-02-21 参照)

[125] Bulmus, A., Freiwald, A., & Wunderlich, C. (2017). Over the Air Software Update Realization within Generic Modules with Microcontrollers Using External Serial FLASH (No. 2017-01-1613). SAE Technical Paper.

[126] ISO22901-1 Road vehicles – Open Diagnostic Data Exchange(ODX) – Part1: Data model specification

[127] ISO13209-2 Road vehicles – Open Test sequence eXchange format(OTX) – Part2:Core data model specification and requirements

[128] ISO13209-3 Road vehicles – Open Test sequence eXchange format(OTX) – Part3:Standard extensions and requirements

[129] Ryu, H. K., Cho, S. R., Piao, S., & Kim, S. H. (2008, July). The design of remote vehicle management system based on OMA DM protocol and AUTOSAR S/W architecture. In Advanced Language Processing and Web Information Technology, 2008. ALPIT'08. International Conference on (pp. 393-397). IEEE.

[130] Zhang, J., Liao, Z., & Zhu, L. (2015). Research on Design and Implementation of Automotive ECUs Software Remote Update. In Applied Mechanics and Materials (Vol. 740, pp. 847-851). Trans Tech Publications.

[131] 野澤優尚, 小沼寛, & 清原良三. (2015). 車載 ECU 向けソフト更新のためのデータ圧縮方式. 研究報告マルチメディア通信と分散処理 (DPS), 2015(4), 1-6.

[132] 小沼寛, 野澤優尚, & 清原良三. (2015).bsdiff を応用した ECU ソフトウェア高速ダウンロード. 情報処理学会第 78 回全国大会

[133] Lee, Y. S., Kim, J. H., Van Hung, H., & Jeon, J. W. (2015, August). A parallel re-programming method for in-vehicle gateway to save software update time. In Information and Automation, 2015 IEEE International Conference on (pp. 1497-1502). IEEE.

[134] Jang, S. J., & Jeon, J. W. (2015, August). Software reprogramming performance analysis of CAN FD and FlexRay protocols. In Information and Automation, 2015 IEEE International Conference on (pp. 2535-2540). IEEE.

[135] Lee, Y. S., Kim, J. H., Jang, S. J., & Jeon, J. W. (2016, January). Automotive ECU Software Reprogramming Method Based on Ethernet Backbone Network to Save Time. In Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication (p. 39). ACM.

[136] 宮本善則他「ECHONET Lite による蓄電池管理システムの開発」日本学術振興会産学協力研究委員会 第 31 回インターネット技術第 163 委員会研究会

[137] TTC TR-1043:ホームネットワーク通信インタフェース実装ガイドライン

[138] ECHONET コンソーシアム「ECHONET 機器認証試験仕様書」 ECHONET Lite 規格 Ver.1.0*用 http://www.echonet.gr.jp

[139] OSGi Alliance「OSGi Service Platform Release4 Device Access Specification Version1.1」http://www.osgi.org/(2019-2-21 参照)

[140] RFC1960 A String Representation of LDAP Search Filters http://www.ietf.org/rfc/rfc1960.txt (2019-2-21 参照)

[141] JASO TP-15002(JP) 自動車の情報セキュリティ分析ガイド

[142] https://www.ipa.go.jp/security/vuln/CVSS.html (2018-02-28 参照)

[143] https://automotive.softing.com/index.php?id=1860&L=4 (2019-02-21 参照)

[144] https://www.openssl.org/ (2018-06-28 参照)

[145] https://www.renesas.com/jp/ja/products/microcontrollers-microprocessors/rh850.html (2019-02-21 参照)

[146] http://www.tessera.co.jp/fl/f1l-176.html (2019-02-21 参照)

[147] https://jp.vector.com/vj_vn1600_jp.html (2018-06-28 参照)

[148] https://jp.vector.com/vj_xl_driver_library_jp.html (2018-06-28 参照)

[149] http://www.hitachi-solutions.co.jp/company/press/news/soft/archive2008/news496.html (2018-02-26 参照)

[150] http://www.oracle.com/technetwork/java/embedded/documentation/me-e-otn-faq-1852008.pdf (2018-06-28 参照)

[151] http://www.bzip.org/(2019-02-21 参照)

[152] Infineon TriCore Brochure (Date: 09 / 2016)

[153] TC1797 Data Sheet V1.3 2014-08 https://www.infineon.com/dgdl/TC1797_DS_V1%203.pdf

[154] Ultra-ReliableMPC574xP MCU (Panther) forAutomotive & Industrial SafetyApplications

[155] MPC5744P Data Sheet https://www.nxp.com/docs/en/data-sheet/MPC5744P.pdf

[156] Burrows, M., & Wheeler, D. (1994). A block-sorting lossless data compression algorithm. In DIGITAL SRC RESEARCH REPORT.

[157] Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. Proceedings of the IRE, 40(9), 1098-1101.

[158] https://tools.ietf.org/html/rfc1951(2019-02-21 参照)

[159] http://www.7-zip.org/sdk.html(2019-02-21 参照)

[160] Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. IEEE Transactions on information theory, 23(3), 337-343.

[161] Martin, G. N. N. (1979, July). Range encoding: an algorithm for removing redundancy from a digitised message. In Proc. Institution of Electronic and Radio Engineers International Conference on Video and Data Recording.

[162] TOPPERS ATK2 https://www.toppers.jp/atk2.html (2019-02-21 参照)

業績リスト

1. 査読付きの学術論文誌

1.1. 本論文に関わる論文

寺岡秀敏, 山﨑裕紀, 櫻井康平, 船迫陽介, & 尾崎友哉. (2018). 軽量スクリプト言語を用いた自動車ソフトウェア遠隔更新制御方式の検討. 情報処理学会論文誌コンシューマ・デバイス & システム (CDS), 8(3), 32-42.

寺岡秀敏, 中原章晴, & 黒澤憲一. (2017). 車載 ECU 向け差分更新方式. 情報処理学会論文誌コンシューマ・デバイス & システム (CDS), 7(2), 41-50.

寺岡秀敏, 今井光洋, 小坂忠義, 奈良祐樹, & 小田輝. (2013). サービスゲートウェイ向け ECHONET Lite バンドルの開発. 情報処理学会論文誌コンシューマ・デバイス & システム (CDS), 3(3), 20-29.

1.2. その他の論文

木谷光博, 片岡幹雄, & **寺岡秀敏**. (2016). 自動運転向け車内ネットワークシステムにおけるデータ伝送方式の開発. 情報処理学会論文誌コンシューマ・デバイス & システム (CDS), 6(2), 43-51.

宮田克也, **寺岡秀敏**, 関原拓也, & 芳野明彦. (2013). ホームゲートウェイ向け機器管理制御フレームワークの開発. 情報処理学会論文誌コンシューマ・デバイス & システム (CDS), 3(3), 39-48.

2. 査読付きの国際会議

Teraoka, H., Nakahara, F., & Kurosawa, K. (2017, October). Incremental update method for vehicle microcontrollers. In Consumer Electronics (GCCE), 2017 IEEE 6th Global Conference on (pp. 1-2). IEEE.

Teraoka, H., Nakahara, F., & Kurosawa, K. (2016, October). Incremental update method for resource-constrained in-vehicle ECUs. In Consumer Electronics, 2016 IEEE 5th Global Conference on (pp. 1-2). IEEE.

Balaji, B., **Teraoka, H**., Gupta, R., & Agarwal, Y. (2013, November). Zonepac: Zonal power estimation and control via hvac metering and occupant feedback. In Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings (pp. 1-8). ACM.

3. 査読のない国内会議（研究会など）

3.1. 本論文に関わる口頭発表

寺岡秀敏, 山﨑裕紀, 櫻井康平, & 尾崎友哉. (2018). 軽量スクリプト言語を用いた自動車ソフトウェア遠隔更新制御方式の開発. 研究報告コンシューマ・デバイス & システム (CDS), 2018(11), 1-7.

寺岡秀敏, 中原章晴, & 黒澤憲一. (2016). 車載 ECU 向け差分更新方式. 研究報告コンシューマ・デバイス & システム (CDS), 2016(5), 1-8.

寺岡秀敏, 今井光洋, 小坂忠義, 奈良祐樹, & 小田輝. (2013). サービスゲートウェイ向け ECHONET Lite バンドルの開発. 研究報告コンシューマ・デバイス & システム (CDS), 2013(39), 1-8.

3.2. その他の口頭発表

木谷光博, 片岡幹雄, & **寺岡秀敏**. (2016). 自動運転向け車内ネットワークシステムにおけるデータ伝送方式の開発. 情報処理学会論文誌コンシューマ・デバイス & システム (CDS), 6(2), 43-51.

宮田克也, **寺岡秀敏**, 関原拓也, & 芳野明彦. (2013). ホームゲートウェイ向け機器管理制御フレームワークの開発. 研究報告コンシューマ・デバイス & システム (CDS), 2013(40), 1-8.

4. 技術資料

Sakurai, K., Kataoka, M., Kodaka, H., Kato, A., **Teraoka, H.**, & Kiyama, N. Connected Car Solutions Based on IoT.

櫻井康平, 片岡幹雄, 小高浩, 加藤淳, **寺岡秀敏**, & 木山昇. (2017). IoT 技術を活用したコネクテッドカーソリューション (Driving Forward with Future Vehicles 安全に, 快適に, 環境と共生するクルマ社会へ)--(「つながるクルマ」 で実現する自動運転技術). 日立評論, 99(5), 514-519.

Teraoka, H., Balaji, B., Zhang, R., Nwokafor, A., Narayanaswamy, B., & Agarwal, Y. (2014). BuildingSherlock: Fault Management Framework for HVAC Systems in Commercial Buildings. Department of Computer Science and Engineering, University of California, San Diego.